БРАЙАН КЕРНИГАН

UNIX

```
long epoch = System.currentTimeMillis()/1000;
epoch = (DateTime.Now.ToUniversalTime().Ticks
- 621355968000000000) / 10000000;   SELECT unix
timestamp(now())   Math.round(new Date().get-
Time()/1000.0) getTime()   date +%s
```

ВРЕМЯ UNIX®

A History and a Memoir

UNIX

A History and a Memoir

Brian Kernighan

Kindle Direct Publishing

БРАЙАН КЕРНИГАН

UNIX

ВРЕМЯ UNIX®

A History and a Memoir

ПИТЕР®
Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону • Самара • Минск

2021

ББК 32.973.2-018.2
УДК 004.451
К36

Керниган Брайан

К36 Время UNIX. A History and a Memoir. — СПб.: Питер, 2021. — 224 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1669-0

Операционная система Unix завоевала мир, чего совсем не могли представить ее создатели, творя историю на последнем этаже Bell Labs в 1969 году. Идея этой ОС оказала колоссальное влияние на разработку программного обеспечения и развитие операционных систем. Вы узнаете о том, как зарождалась система Unix, чем она примечательна и почему занимает столь важное место в компьютерном мире, а также об удивительных людях, вложивших в нее силы и душу. Книга представляет собой честный и остроумный рассказ о жизни айтишного сообщества тех времен — никакой зауми, страниц кода и ссылок. Расслабьтесь и получайте удовольствие, погрузившись в историю, полную приключений и открытий.

Брайан Керниган — автор и соавтор дюжины книг по программированию, включая легендарные «Язык программирования Си» и «Unix. Программное окружение». На протяжении 30 лет был членом той самой группы UNIX в исследовательском центре Bell Labs и не только наблюдал создание UNIX.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.2
УДК 004.451

ISBN 978-1695978553 англ.

ISBN 978-5-4461-1669-0

КРАТКОЕ СОДЕРЖАНИЕ

ОГЛАВЛЕНИЕ

Памяти Денниса Ритчи (DMR)

ПРЕДИСЛОВИЕ

> В старых воспоминаниях приятно то, что их покрывает розовая дымка времени. В памяти фиксируется все хорошее и то, что прошло проверку временем, а также радость от участия в создании того, что сделало жизнь лучше.
>
> *Из статьи Денниса Ритчи*
> *The Evolution of the Unix Time-sharing System,*
> *октябрь 1984*

С момента появления оперативной системы Unix на последнем этаже Bell Labs в 1969 году она распространилась так широко, как и не могли мечтать ее создатели. Она вызвала разработку инновационного программного обеспечения, повлияла на множество программистов и изменила путь развития компьютерных технологий.

Не слишком известные за пределами технического сообщества операционная система Unix и ее производные лежат в основе множества всем знакомых систем. Google, Facebook, Amazon и многие другие сервисы работают на Linux — Unix-подобных ОС, о которых я расскажу позже. Если у вас есть мобильный телефон или компьютер Mac, знайте, что там установлена какая-то версия Unix. Системы семейства Unix обеспечивают работу виртуальных помощников, таких как Alexa, и навигационного программного обеспечения. За всей той рекламой, которой вас засыпают всякий раз при просмотре веб-страниц, тоже стоят системы семейства Unix. Скорее всего, там реализованы механизмы отслеживания действий пользователей, позволяющие сделать рекламу более релевантной.

Первая ОС Unix была создана более 50 лет назад двумя людьми при поддержке небольшой группы коллег. Мне повезло присутствовать при этом, хотя лично я и не имею отношения к этому процессу. Максимум, чем я могу похвастаться, это немного полезного программного обеспечения, а также написанные вместе с первоклассными соавторами книги, позволяющие больше узнать о Unix, ее языках программирования, инструментах и философии.

Эта книга стала коктейлем из истории и мемуаров, моим взглядом на процесс зарождения Unix и попыткой объяснить, что это за операционная система, как она появилась и почему приобрела большое значение. Это не научная работа, здесь нет сносок, и, более того, книга получилась менее исторической и более мемуарной, чем я планировал.

Книга написана для всех, кто интересуется компьютерными системами или историей изобретений. В ней присутствует технический материал, но я постарался дать объяснения, позволяющие оценить основные идеи и понять их важность даже людям, далеким от технологий. Впрочем, все, что кажется слишком сложным, можно пропустить — читать каждое слово необязательно. С другой стороны, тем, кто имеет опыт программирования, некоторые объяснения могут показаться очевидными и чрезмерно упрощенными, но я надеюсь, что и такие читатели сочтут полезными рассказы о том, как возникали те или иные идеи.

Я пытался быть насколько возможно точным, но человеческая память несовершенна. Оказалось, что интервью, личные воспоминания коллег, устные истории, книги и документы, на которые я опирался при написании этой книги, не всегда совпадали с моими воспоминаниями. Противоречивой иногда оказывалась и информация из разных источников, повествующих о том, что кто и когда сделал.

К счастью, многие из участников описываемых событий пока живы и помогли мне внести поправки. Разумеется, они тоже могут помнить не все или помнить что-то не так. Они тоже могут страдать от провалов в памяти или носить розовые очки, но любые ошибки, которые остались в книге, — моя вина, по крайней мере до тех пор, пока я не смогу их на кого-то переложить.

Цель написания этой книги — рассказать замечательную историю самого продуктивного и созидающего периода в истории информационно-вычис-

лительных систем. Важно понять, как эволюционировали привычные для нас технологии. Решения, определявшие пути развития технологий, принимались в условиях давления и временных ограничений. Чем лучше мы узнаем историю, тем выше сможем оценить гениальность людей, которые смогли создать Unix, а возможно, и лучше поймем, почему современные компьютерные системы именно такие, какие есть. Я покажу, что действия, которые с современной точки зрения могут показаться неправильными или ошибочными, часто оказывались естественным следствием господствовавшего в те времена понимания тех или иных вещей, а также ограниченности доступных ресурсов.

Моя история не только о Unix, хотя все повествование строится вокруг нее. Но я расскажу и про язык программирования Си — один из наиболее используемых языков, на котором написаны системы, управляющие интернетом, и множество служб. После появления Unix в Bell Labs возникли и другие языки, в том числе получивший широкое распространение C++. На этом языке написаны такие приложения пакета Microsoft Office, как Word, Excel или Powerpoint, а также большинство знакомых вам браузеров. Пара десятков инструментов, которыми программисты пользуются каждый день и даже не представляют себе, как можно обходиться без них, были написаны практически сразу после появления Unix. При этом сегодня они так же актуальны, как и 40–50 лет назад.

Важную роль играет и теория компьютерных наук (computer science), зачастую позволяющая реализовывать очень удобные инструменты. Изучение аппаратного обеспечения приводило к появлению инструментов проектирования, интегральных микросхем, различных вариантов архитектуры вычислительных систем и необычных устройств специального назначения. Благодаря комбинации всего этого часто рождались неожиданные изобретения. Именно она обеспечила продуктивность Bell Labs в самых разных областях.

Кроме того, история возникновения технологических инноваций всегда интересна и актуальна. Bell Labs, где появилась Unix, не просто послужили колыбелью для множества хороших идей, но и смогли извлечь из них выгоду. Они стали источником меняющих мир изобретений, и мы можем многому научиться, внимательно рассмотрев, как все это происходило.

История Unix наглядно показывает, как нужно проектировать и создавать программное обеспечение и как эффективно использовать компьютеры.

В своем рассказе я попытался подчеркнуть эти моменты. К примеру, философия Unix позволила решить множество задач объединением существующих программ. Такой подход избавлял от необходимости писать новое программное обеспечение. Фактически это давно известная стратегия: разделяй и властвуй. Крупную задачу можно разбить на ряд более мелких и проще решаемых, а полученные фрагменты объединить неожиданным образом.

Операционная система Unix оказалась наиболее заметным программным обеспечением, созданным в Bell Labs, но это далеко не единственный их вклад в информационные технологии. Исследовательский центр компьютерных наук — легендарное подразделение 1127 — был необычайно продуктивным в течение двух или трех десятилетий. Его деятельность обусловливалась операционной системой Unix, которая служила основой, но достигнутые результаты выходят далеко за пределы Unix. Книги, написанные сотрудниками 1127, много лет служили базовыми учебниками по информатике и справочниками для программистов. Подразделение 1127 было влиятельной научно-исследовательской лабораторией, одной из наиболее продуктивных как в то время, о котором пойдет речь, так и позднее.

Почему Unix и все, что было с ней связано, ждал такой успех? Как эксперимент двух человек вырос в проект, в буквальном смысле слова изменивший мир? Был ли это уникальный случай, настолько редкий, что ничего подобного больше не повторится? Ответ на вопрос, можно ли спланировать такие колоссальные результаты, я дам в конце книги. Хотя мне кажется, что своим успехом Unix обязана случайному сочетанию факторов: два исключительно талантливых человека, отличный вспомогательный персонал, талантливое и компетентное руководство, стабильное финансирование в корпорации, в которой умели работать на перспективу, и отсутствие препятствий для любой исследовательской деятельности. Кроме того, внедрению новой операционной системы способствовало быстрое развитие технологий, в процессе которого размеры оборудования уменьшались, а оно дешевело при экспоненциальном росте мощности.

Первые годы работы над Unix для меня и многих других сотрудников Bell Labs были поразительно продуктивными и веселыми. Я надеюсь, эта книга сможет передать вам ощущение радости творчества и счастья от того, что мы делаем жизнь лучше, как говорится в эпиграфе Денниса Ритчи в начале этого раздела.

БЛАГОДАРНОСТИ

При написании этой книги я испытал неожиданную радость воссоединения со многими друзьями и коллегами, которые щедро поделились своими воспоминаниями и замечательными историями. Сложно передать ценность полученной от них информации. Я не смог включить в книгу все истории, но испытал огромное удовольствие, слушая их. Я в долгу перед многими замечательными людьми, с которыми мне посчастливилось работать.

Биографические материалы разбросаны по всей книге, причем их основная часть посвящена трем людям, без которых не случилось бы Unix, — Кену Томпсону, Деннису Ритчи и Дугу Макилрою. Кен и Дуг дали мне бесценные отзывы о книге. Разумеется, они никоим образом не несут ответственности за сведения, в которых я ошибся или которые непреднамеренно исказил. Чрезвычайно важные комментарии и советы я получил и от братьев Денниса — Джона и Билла. Его племянник Сэм подробно прокомментировал несколько черновиков.

Как он много раз делал это в прошлом, Джон Бентли поделился со мной бесценными идеями, дал полезные советы по структурированию материала и расстановке акцентов, рассказал множество историй и подробно разобрал по крайней мере с полдюжины черновиков. Я в очередной раз в огромном долгу перед Джоном.

Кроме ценных советов Джерард Хольцманн предоставил мне архивные материалы и оригинальные фотографии, которые помогли сделать книгу визуально насыщенной. Пол Керниган прочитал несколько черновиков и нашел множество опечаток. Он также предложил несколько отличных названий, хотя я, к сожалению, так и не решился использовать вариант «История людей, говорящих на Unix».

Свою критику и рассказы о периоде становления Unix мне предоставили Аль Ахо, Майк Бьянки, Стю Фельдман, Стив Джонсон, Майкл Леск, Джон Линдерман, Джон Мэши, Петер Нойман, Роб Пайк, Ховард Трики и Питер Вайнбергер. Многое из этого материала я цитировал или перефразировал.

Множество полезных комментариев и сведений я получил и от Майкла Бачанда, Дэвида Брока, Грейс Эмлин, Майи Хамин, Билла Джоя, Марка Кернигана, Мег Керниган, Уильяма Макграта, Питера Макилроя, Арнольда

Роббинса, Ионы Синовица, Бьёрна Страуструпа, Уоррена Туми и Джанет Вертеси. Я глубоко благодарен за оказанную мне щедрую помощь и готов лично нести ответственность за любые ошибки или неверные толкования. За последние 50 лет значительный вклад в Unix внесли и многие другие, и я приношу извинения всем, чью работу не упомянул.

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.

ГЛАВА 1

BELL LABS

Одна политика, одна система, универсальный сервис.

Миссия компании AT&T, 1907 год

При беглом осмотре расположенные в удивительной сельской местности Телефонные Лаборатории Белла напоминали большую современную фабрику; да они ею и были. Но это фабрика идей, производственные линии которой невидимы глазу.

Цитата из книги Артура Кларка «Голос через океан» 1974 года, вставленная в книгу Джона Гертнера «Фабрика идей», 2012 год

Чтобы понять, как появилась идея Unix, нужно понять, как работали Bell Labs, и прочувствовать существовавшую там творческую среду.

Американская телефонно-телеграфная компания AT&T появилась путем слияния множества телефонных компаний из разных штатов. Вскоре после ее создания стало понятно, что для решения научных и инженерных проблем, с которыми сталкивалась компания в попытках создать национальную телефонную сеть, требуется отдельное подразделение. Так в 1925 году появилась дочерняя компания, занимающаяся исследованиями и разработками, — Bell Telephone Laboratories (Телефонные Лаборатории

Белла). Ее название регулярно сокращали до Bell Labs, BTL или просто Labs, но главной задачей всегда оставалась телефония.

Первоначально она располагалась в Нью-Йорке по адресу 463 West Street, но после начала Второй мировой войны AT&T стала предоставлять военным экспертные услуги по системам связи, компьютерному управлению зенитными орудиями, радарам и криптографии. Эта работа частично осуществлялась в Нью-Джерси, в 33 км западнее Нью-Йорка, в районе Мюррей-Хилл, который был частью небольших городов Нью-Провиденс и Беркли-Хайтс.

Эти локации отмечены точками на карте (рис. 1.1). Первоначально компания находилась на берегу Гудзона, недалеко от выезда на шоссе 9A. В Мюррей-Хилл Bell Labs оказались на границе городов Нью-Провиденс и Беркли-Хайтс, к северу от магистрали 78.

Постепенно деятельность Bell Labs все сильнее концентрировалась в Мюррей-Хилл, и к 1966 году состоялся полный переезд. В результате в Мюррей-Хилл оказалось более 3000 человек, из которых по крайней мере тысяча обладали докторскими степенями в области физики, химии, математики и различных инженерных специальностей.

На рис. 1.2 вы видите аэрофотоснимок комплекса в Мюррей-Хилл в 1961 году. Тогда существовало три основных здания. Первое — в правом нижнем углу, второе — в верхнем левом, третье — квадратное с открытым внутренним двором. До появления в 1970-х годах еще двух зданий был 400-метровый проход из первого здания во второе.

Во втором здании я провел более 30 лет, с момента прихода на стажировку в 1967-м до ухода на пенсию в 2000-м. На снимке точками отмечены мои кабинеты, занимавшие два крыла на пятом (верхнем) этаже. В дальнем конце здания находилась лестница 9, а лестница 8 была на одно крыло ближе к центру. Между этими лестницами, на шестом этаже, в мансарде, и располагалась комната Unix.

На рис. 1.3 предоставленное Google спутниковое изображение Лабораторий в 2019 году. Здания 6 (внизу слева, с пометкой) и 7 (вверху справа) появились в начале 1970-х годов. С 1996 года несколько лет в здании 6 располагалась штаб-квартира компании Lucent Technologies. Обратите внимание, как на присваиваемых Google ярлыках отражается корпоративная история. Здание, помеченное как Bell Labs, и въездная дорога называются Alcatel-Lucent Bell Labs, выезд — Lucent Bell Labs, а пирамидальную крышу административного корпуса в здании 6 украшает пометка Nokia Bell Labs.

Рис. 1.1. Из Нью-Йорка в Мюррей-Хилл, Нью-Джерси

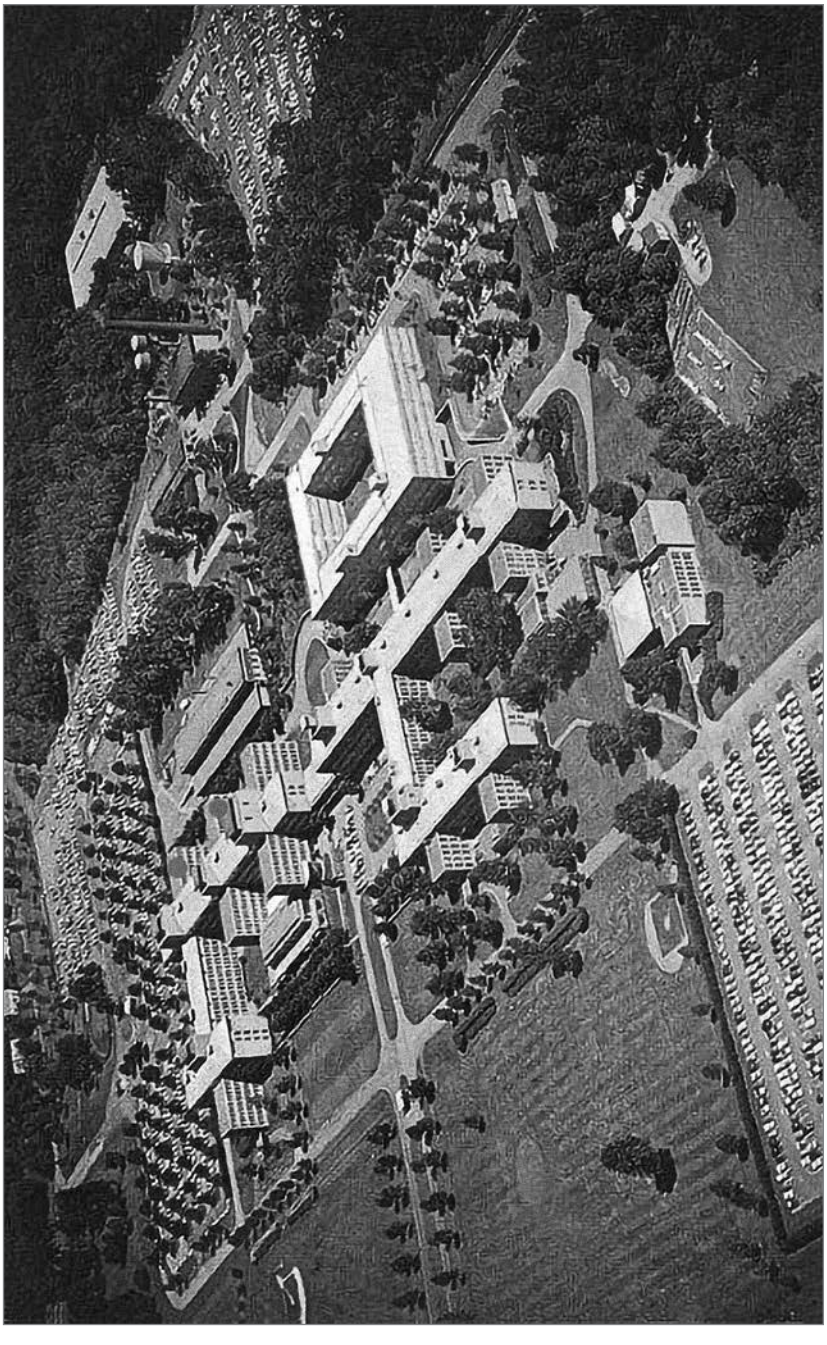


Рис. 1.2. Bell Labs в 1961 году (из архива Bell Labs)

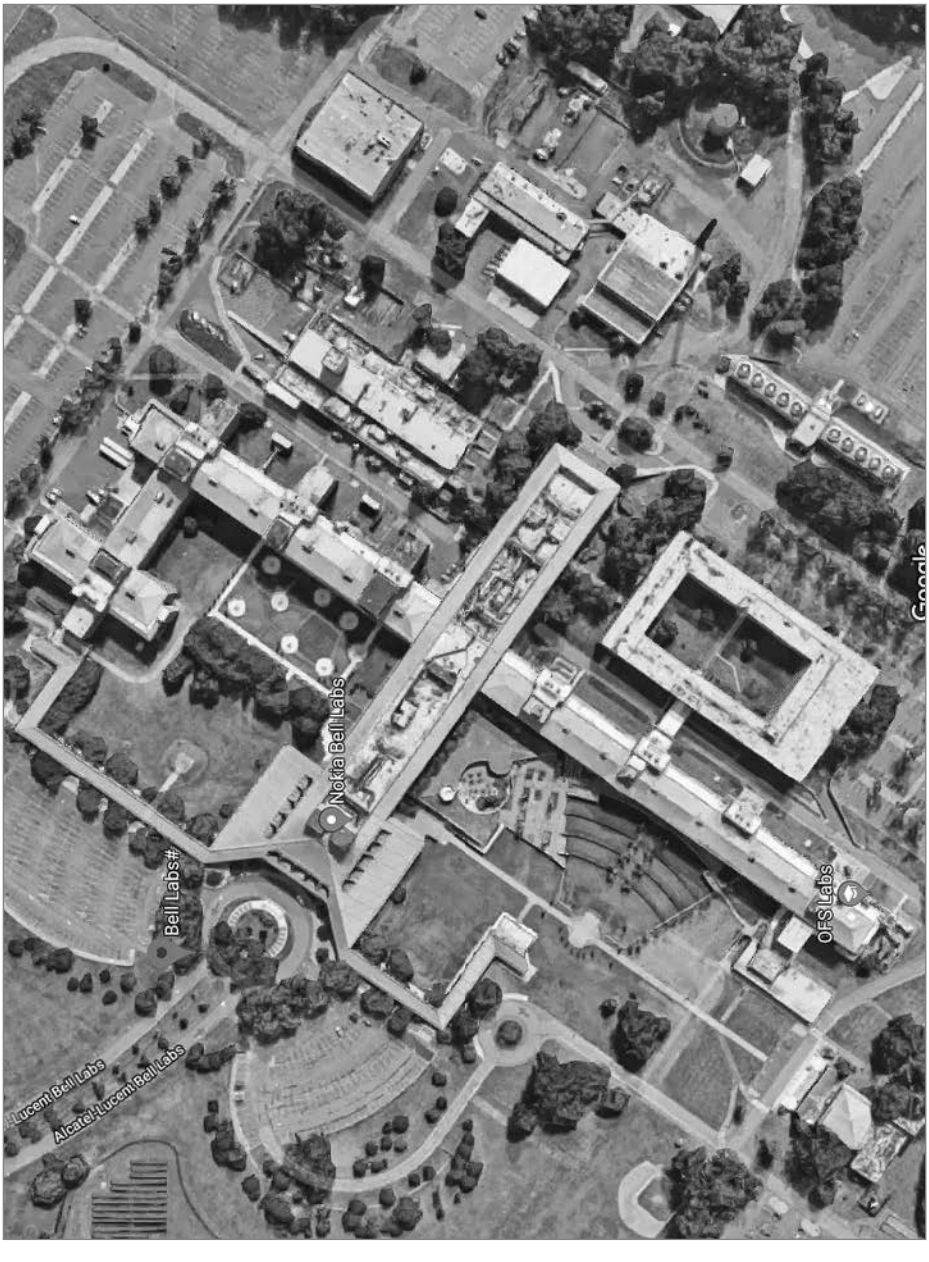


Рис. 1.3. Bell Labs в 2019 году; здание 6 внизу слева

Мне не хватит знаний для изложения подробной истории Bell Labs, более того, уже есть люди, которые ее написали. Особенно мне нравятся посвященная естественным наукам «Фабрика идей» Джона Гертнера и «Информация» Джеймса Глейка, отлично описывающая, что происходило в области информатики. Интерес представляет и огромное (семь томов и почти пять тысяч страниц) официальное издание от Bell Labs *A History of Science and Engineering in the Bell System*.

1.1. ЕСТЕСТВЕННЫЕ НАУКИ В BELL LABS

Сначала большинство исследований в Bell Labs проводилось в областях физики, химии, материаловедения и систем связи. Сотрудники могли сами выбирать тему исследований, ведь актуальных задач было так много, что не составляло труда найти что-то интересное и одновременно полезное как для Bell System, так и для всего мира.

Научно-технические достижения Bell Labs изменили мир. Первым из них стал транзистор, изобретенный в 1947 году Джоном Бардином, Уолтером Браттейном и Уильямом Шокли, которые пытались совершенствовать усилители для междугородных телефонных линий. Транзистор появился в результате фундаментальных исследований свойств полупроводниковых материалов при поиске менее энергоемких и более устойчивых физических приборов, чем радиолампы, которые в 1940-х годах безальтернативно применялись для изготовления коммуникационного оборудования и, между прочим, стали элементной базой компьютеров первого поколения.

В 1956 году изобретение транзистора было отмечено Нобелевской премией по физике. Это одна из девяти Нобелевских премий, присужденных за работу, хотя бы частично выполненную в Bell Labs. Другие важные изобретения — это усилители с отрицательной обратной связью, солнечные элементы, лазеры, мобильные телефоны, спутники связи и приборы с зарядовой связью (которые обеспечивают работу встроенных в телефоны камер).

По приблизительным оценкам, в 1960–1980-х исследованиями в Bell Labs (в основном в Мюррей-Хилл) занималось 3000 человек. Кроме того,

от 15 000 до 25 000 принадлежали к различным группам разработчиков оборудования и систем для Bell System. Кто оплачивал их труд?

По сути, компания AT&T обладала монополией на предоставление телефонной связи в большей части США. При этом цены на ее услуги регулировались федеральными и государственными органами, кроме того, компания не могла заниматься бизнесом, не имеющим прямого отношения к предоставлению телефонной связи.

Такой режим регулирования применялся много лет. Услуги компании AT&T должны были предоставляться любому (всеобщий охват), независимо от сложности доступа и выгодности. В качестве компенсации компания имела стабильную и предсказуемую общую норму прибыли.

В рамках этой договоренности часть своего дохода AT&T направляла в Bell Labs с целью дальнейшего совершенствования услуг связи. По сути, Лаборатории оплачивались скромным налогом на все телефонные звонки в стране. Согласно документу А. Майкла Нолла, около 2,8% своих доходов AT&T тратила на исследования и разработки, а около 0,3% — на фундаментальные исследования. Я не знаю, будет ли такая схема работать сегодня, но десятилетиями эта договоренность порождала как постоянные улучшения в системе телефонной связи, так и значительное количество фундаментальных научных открытий.

Определяющим фактором было стабильное финансирование. Компании AT&T оно обеспечивало долгосрочную перспективу, а сотрудникам Bell Labs давало возможность заниматься исследованиями, не приносящими быстрой отдачи. В современном мире ничего подобного нет — при планировании зачастую не заглядывают дальше нескольких месяцев, зато масса усилий тратится на обдумывание финансовых показателей в следующем квартале.

1.2. КОММУНИКАЦИИ И COMPUTER SCIENCE

Bell Labs стали пионером в проектировании, создании и совершенствовании систем связи. Этот общий термин охватывает все — от проектирования бытовых устройств, таких как телефоны, до инфраструктуры коммутационных систем, радиорелейных вышек и оптоволоконных кабелей.

Иногда бывает так, что решение такого спектра практических задач порождает прогресс в фундаментальных науках. Например, в 1964 году Арно Пензиас и Роберт Вильсон пытались выяснить, чем обусловлена избыточная шумовая температура антенны, которую Bell Labs применяли для обнаружения радиосигналов, отражаемых спутниковыми ретрансляторами «Эхо». В конце концов они пришли к выводу, что шум вызывался фоновым излучением, оставшимся от Большого взрыва, давшего начало расширению Вселенной. За это открытие в 1978 году Пензиас и Вильсон получили Нобелевскую премию по физике. Арно прокомментировал это так: «Большинство получает Нобелевскую премию за то, что ищет. Мы же получили ее за то, от чего пытались избавиться».

Другая задача, которую ставили в Bell Labs, — математически описать принципы работы систем связи. В результате Клод Шеннон заложил основы теории информации. Отчасти базой для этого послужила криптография, которую он изучал в годы Второй мировой войны. Статья Шеннона «Математическая теория связи», опубликованная в 1948 году в реферативном журнале компании Bell System, объясняла границы возможностей систем передачи информации и исходные принципы их разработки. В Мюррей-Хилл Шеннон работал с начала 1940-х до 1956 года, затем вернулся в качестве преподавателя в Массачусетский технологический институт (MIT), где в свое время учился в аспирантуре. Умер Шеннон в 2001 году в возрасте 84 лет.

По мере роста компьютерных мощностей и снижения их стоимости компьютеры все чаще стали применять для анализа данных, а также для моделирования и симуляции физических систем и процессов. Компьютерами в Bell Labs стали заниматься с 1930-х годов, а к концу 1950-х там появились большие компьютерные центры.

В начале 1960-х годов была образована группа, специализирующаяся в области компьютерных наук. Туда вошли как сотрудники, ранее занимавшиеся исследованиями в сфере математики, так и сотрудники, которые управляли большим центральным компьютером в Мюррей-Хилл. Новый коллектив получил название Исследовательского центра компьютерных наук (Computing Science Research Center), и хотя еще некоторое время они предоставляли компьютерные услуги всем обитателям Мюррей-Хилл, это была уже не группа обслуживания, а научно-исследовательский отдел.

В 1970 году группа, управляющая компьютерным оборудованием, была преобразована в отдельную организацию.

1.3. BWK[1] В BELL LABS

Этот раздел — по большей части моя личная история, которая, я надеюсь, даст представление о том, как череда счастливых случаев сделала меня программистом в таком идеально подходящем для этого занятия месте, как Bell Labs.

Я учился в университете своего родного города Торонто. В те годы существовала программа «Инженерная физика» (которая позже стала называться «Технические науки»), подходящая для тех, кто не знал, в какой области он хотел бы специализироваться. Я окончил университет в 1964 году, на заре компьютерных технологий: первый компьютер я увидел на третьем курсе. Это был огромный IBM 7094, единственный компьютер на весь университет, передовое чудо техники. Он имел 32 Кбайт памяти (32 768 слов по 36 битов) на магнитных сердечниках (сегодня мы бы сказали, что это 128 Кбайт) и дополнительное хранилище в виде больших механических дисков. В то время такой компьютер стоил 3 миллиона долларов США, устанавливался в большой комнате с кондиционером и обслуживался профессиональными операторами. У обычных людей (и уж тем более у студентов) доступа к нему не было.

Поэтому во время учебы я не имел дела с компьютерами, хотя и пытался учить язык программирования Фортран. Сочувствую всем, кто когда-либо писал свою первую программу. Я прочитал о Фортране II превосходную книгу Дэниеля Маккракена, назубок выучил все правила, но это никак не помогало. Подозреваю, что на подобный концептуальный барьер наталкивались многие.

Летом перед последним годом обучения в колледже я получил работу в компании Imperial Oil, в группе, которая разрабатывала программное обеспечение для оптимизации нефтеперерабатывающих заводов. Компа-

---

[1] Bwk называли оригинальную версию языка Awk, которую написал и активно поддерживал впоследствии автор книги Брайан Керниган. — *Примеч. пер.*

ния Imperial Oil частично принадлежала корпорации Standard Oil, Нью-Джерси, которая в 1972 году превратилась в корпорацию Exxon.

Честно признаюсь, как стажер я оказался не на высоте. Все лето я писал на языке Кобол (COBOL, COmmon Business Oriented Language) гигантскую программу для анализа данных нефтеперерабатывающего завода. Уже не помню, зачем она требовалась, но точно знаю, что работать она так и не начала. Программировать я попросту не умел. Язык Кобол практически не давал возможности хорошо структурировать программу, да и структурированное программирование на тот момент еще не было изобретено, поэтому мой код состоял из множества операторов `IF`, которые то и дело ветвились, потому что требовалось добавить очередную операцию.

Еще я пытался писать программы на Фортране для мейнфрейма IBM 7010, так как этот язык я знал лучше, чем Кобол, кроме того, он больше подходил для анализа данных. После нескольких недель борьбы с JCL (Job Control Language, именно на этом языке осуществлялось управление заданиями) я пришел к выводу, что на нашем 7010 отсутствует компилятор Фортрана. Сообщения об ошибках, которые выдавал компилятор, были настолько непостижимыми, что их не мог понять никто.

Несмотря на разочарования этого лета, мой интерес к компьютерам не угас. Хотя официально в университете не было курсов по информатике, для дипломной работы я выбрал такую крайне популярную в те годы тему, как искусственный интеллект. Тогда казалось, что достаточно толики программирования, и мы сможем доказывать теоремы, играть с машиной в шахматы и шашки и переводить с одного языка на другой.

В 1964 году, завершив свое обучение, я понятия не имел, что делать дальше, и поэтому, как и многие студенты, решил поступать в аспирантуру. Разослал заявления в полдюжины американских заведений (в то время в Канаде подобное не очень практиковалось), и мне повезло. Меня приняли в несколько вузов, в том числе в MIT и в Принстон. В Принстоне докторскую степень обычно получали за три года, в то время как в MIT это занимало семь лет. Принстон предложил полную стипендию, а в MIT мне пришлось бы по 30 часов в неделю отрабатывать лаборантом. Выбор был очевидным, тем более в Принстоне уже учился мой хороший друг Аль Ахо, который был старше меня на год. Это решение оказалось невероятно удачным.

В 1966 году мне снова повезло. Я попал на летнюю стажировку в MIT. Это произошло отчасти благодаря тому, что годом ранее там очень результативно поработал другой аспирант из Принстона, Ли Вариан. Тем летом я писал на MAD (Michigan Algorithm Decoder, диалект языка Algol 58) в системе разделения времени (Compatible Time-Sharing System, CTSS) инструменты для новой операционной системы Multics. О ней мы подробно поговорим в главе 2. Вообще, конечно, она называлась MULTICS, но запись в нижнем регистре не так раздражает глаза, как и Unix вместо UNIX. Так что я буду и дальше использовать такой не совсем исторически корректный вариант написания.

Номинально моим руководителем в MIT был профессор Фернандо Корбато, которого все называли Корби. Истинный джентльмен, создатель программного пакета CTSS и человек, отвечающий за Multics. В 1990-м он получил премию Тьюринга за фундаментальную работу над системами разделения времени. Корби умер в июле 2019 года в возрасте 93 лет.

Помимо разработки и внедрения CTSS и Multics Корби изобрел пароли для доступа к компьютеру. В пакетных вычислениях в то время не было особой необходимости, зато требовался механизм безопасности для защиты личных файлов на компьютерах с разделением времени и общей файловой системой.

Я до сих пор помню оба автоматически сгенерированных для меня пароля, которые мне выдали на время практики. Второй пароль потребовался, потому что однажды в летней суете файл с паролями оказался в файле с сообщением дня. Пароли хранились в незашифрованном виде, так что их могли видеть все, кто входил в систему.

Помню, как единолично пользовался кабинетом, который принадлежал профессору Джозефу Вейценбауму, так как летом его не было. В начале 1966 года Вейценбаум написал программу «Элиза», которая пародировала диалог с психотерапевтом. Это была одна из первых попыток использовать естественный язык для взаимодействия с компьютером. Программа произвела огромное впечатление, особенно в первый раз, и мне нравилось показывать ее посетителям. Сегодня в Сети есть множество подобных программ, и с ними по-прежнему интересно поиграть, хотя их слабые стороны быстро становятся очевидными.

```
Элиза: Пожалуйста, скажи, что тебя беспокоит?
Я:     Я пытаюсь написать книгу про Unix.
Элиза: Ты пришел ко мне, потому что пытаешься написать книгу про Unix?
Я:     Да.
Элиза: Ты уверен?
Я:     Да.
Элиза: Да, кажется, ты действительно уверен.
Я:     Да.
Элиза: Понятно.
```

Летом 1967 я вытянул счастливый билет: стажировку в Bell Labs, в Исследовательском центре компьютерных наук. Я начал работать на Дуга Макилроя (рис. 1.4). Дуг предложил мне заняться оценкой распределителей памяти. Это был один из его долгосрочных проектов. В лучших традициях стажеров я действовал невпопад и в конце концов решил совсем другую задачу — создал библиотеку функций, облегчающих обработку списка в программах на Фортране. В то лето я плотно программировал на ассемблере для мейнфрейма GE 635, который представлял собой в чистом виде компьютер IBM 7094. В то же время это была упрощенная версия GE 645, специально под операционную систему Multics. Наверное, это был последний раз, когда я писал на ассемблере. И хотя мой код содержал множество ошибок, процесс мне настолько понравился, что я увлекся программированием.

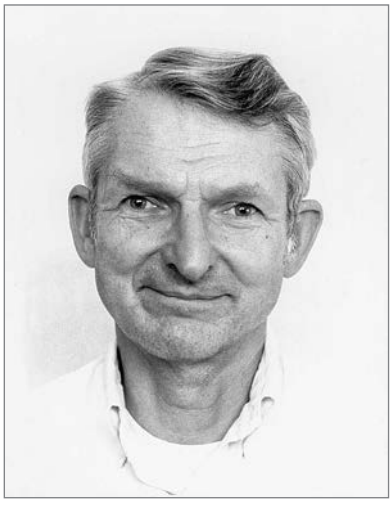


Рис. 1.4. Дуг Макилрой, примерно 1981 год
(из архива Джерарда Хольцманна)

1.4. РАБОЧЕЕ ПОМЕЩЕНИЕ

Иногда судьбу определяет география.

Как стажер я получил кабинет на пятом этаже здания 2 рядом с лестницей 8. В те старые добрые времена даже у стажеров были личные кабинеты. В первый рабочий день я сидел там и не знал, что делать. В 11 утра в дверях возник мужчина и сказал: «Привет, я Дик (далее неразборчиво). Пойдем пообедаем».

Я не помню, как прошел этот обед, зато помню, что потом, когда Дик с неразборчивой фамилией куда-то ушел, я отправился читать табличку с фамилией на двери его кабинета. Ричард Хэмминг! Мой дружелюбный сосед оказался знаменитостью, изобретателем самоконтролирующихся и самокорректирующихся кодов и автором учебника по методам численного анализа, по которому я в то время учился.

Дик (рис. 1.5) стал моим хорошим другом. Он был человеком твердых убеждений и не боялся их высказывать, что, как я думаю, отталкивало некоторых людей, но я наслаждался его компанией и много лет с пользой применял его советы. Он был начальником отдела, где не было сотрудников, что казалось странным. По словам Дика, он приложил массу усилий для достижения подобной комбинации, то есть достойной должности без сопутствующей ответственности. Оценить это я смог, только став начальником отдела с десятком подчиненных.

Летом 1968 года он узнал, что получил премию Тьюринга, которая в сфере информационных технологий считается эквивалентом Нобелевской премии. В то время Нобелевская премия составляла 100 000 долларов, а премия Тьюринга — 2000, так что Дик язвительно заметил, что получил 2 % Нобелевской премии. Это была третья премия Тьюринга; первые две получили другие пионеры компьютерных технологий, Алан Перлис и Морис Уилкс. Дик был номинирован на нее за работы в области численных методов, систем автоматического кодирования, кодов определения и корректировки ошибок.

Благодаря Дику я начал писать книги. Он был невысокого мнения о большинстве программистов, считая, что они плохо подготовлены, если подготовлены вообще. Я до сих пор слышу, как он говорит:

Мы даем им словарь и грамматические правила и говорим: «Теперь ты отличный программист, малыш».

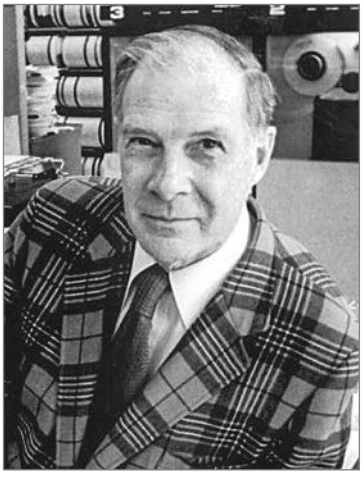


Рис. 1.5. Дик Хэмминг, примерно 1975 год, в своем фирменном клетчатом пиджаке (Википедия)

Он чувствовал, что программированию нужно учить, как и написанию текстов. Должно быть чувство стиля, отделяющее плохой код от хорошего. А программистов следует учить писать код и ценить хороший стиль.

Мы не сошлись во мнении о том, как этого можно достичь, но идея была здравой и легла в основу моей первой книги «Элементы стиля программирования», которую я написал в 1974 году в соавторстве с Пи Джей «Биллом» Плоджером. В те годы он занимал соседний офис. Мы подражали языковому руководству Странка и Вайта «Элементы стиля», приводя плохие примеры и объясняя, как улучшить каждый из них.

Первый пример был взят из книги, которую когда-то показал мне Дик. Он принес текст по численному анализу, возмущенный тем, насколько плохо тот был написан. Я увидел только ужасный фрагмент на Фортране:

```
    DO 14 I=1,N
    DO 14 J=1,N
14 V(I,J)=(I/J)*(J/I)
```

Для тех, кто незнаком с этим языком, поясню. Два вложенных цикла `DO` реализуют операцию, указанную в строке с меткой 14. При этом каждый меняет значение переменной цикла от `1` до `N`. Переменная `V` представляет собой массив из `N` строк и `N` столбцов. Переменная цикла `I` осуществляет перебор строк, а переменная цикла `J` — столбцов.

В результате появляется матрица размером `N` на `N` с 1 на главной диагонали. Все остальные ее элементы равны нулю. Например, для `N` = 5 получим:

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

Принцип работы этого кода основан на том, что в Фортране при целочисленном делении отбрасывается дробная часть. Соответственно, при равенстве переменных `I` и `J` будет получаться единица, во всех остальных случаях — ноль. Такой подход показался мне заумным и неоправданно сложным.

Этот код можно сделать более простым и очевидным. На каждой итерации внешнего цикла внутренний цикл присваивает всем элементам строки `I` значение `0`, а затем внешний цикл присваивает диагональному элементу `V (I, I)` значение 1:

```
C   СОЗДАЕМ ЕДИНИЧНУЮ МАТРИЦУ V
        DO 14 I = 1,N
            DO 12 J = 1,N
    12          V(I,J) = 0.0
    14      V(I,I) = 1.0
```

Из этого выводим первое правило оформления кода:

Пишите ясно, не умничайте.

В 1976 году Дик уволился из Bell Labs и перешел в Школу повышения квалификации офицерских кадров ВМС США в Монтерее, штат Калифорния. Там он преподавал до самой смерти. Он скончался в начале 1998 года в возрасте 82 лет. История гласит, что один из своих курсов он назвал

«Hamming on Hamming» («Хэмминг на Хэмминге»), так как «материал курса появился благодаря собственному опыту, наблюдениям и начитанности». Я могу сказать то же самое про эту главу.

Дик все время тщательно продумывал, что и почему делает. Он повторял, что «цель вычислений — понять суть, а не получить цифры». У него даже был галстук с этой фразой (на китайском языке). Он считал, что со временем половина работы в Bell Labs будет выполняться компьютерами. Никто из его коллег не был согласен с этим прогнозом, но вскоре оказалось, что Дик даже поскромничал в своих оценках.

Он считал, что по пятницам во второй половине дня в голову приходят замечательные мысли, поэтому садился и думал, хотя таким посетителям, как я, был рад в любое время.

Через несколько лет после выхода на пенсию Дик выступил с очень занимательной речью, в которой советовал, как сделать успешную карьеру. Эту речь до сих пор можно найти в Интернете под названием «Вы и ваши исследования» («You and Your Research»). Первое выступление состоялось в марте 1986 года в компании Bell Communications Research; Кен Томпсон отвез меня туда, чтобы мы могли послушать его. Я уже много лет рекомендую всем студентам почитать стенограмму или посмотреть одну из версий видео.

Напротив моего кабинета летом 1967 года находился кабинет еще одного невероятно умного и талантливого программиста — Виктора Высотского (рис. 1.6). В Bell Labs Вик отвечал за подразделение, занимавшееся созданием операционной системы Multics, и был партнером Корби, но при этом умудрялся выделять время на почти ежедневные разговоры со стажером. Вик заставил меня преподавать курс Фортрана физикам и химикам, которым нужно было научиться программировать. Опыт преподавания оказался увлекательным. Он помог мне преодолеть страх перед публичными выступлениями и впоследствии провести множество лекций.

Вскоре после этого Вик переехал в другой отдел Bell Labs, где работал над системой противоракетной обороны Safeguard. Но в конце концов все равно вернулся в Мюррей-Хилл и стал исполнительным директором, ответственным за исследования в области компьютерных наук. То есть оказался на пару ступеней выше меня.

Рис. 1.6. Вик Высотский, примерно 1982 год (из архива Bell Labs)

Весной 1968 года я начал работать над задачей *разбиения графов*, которую мой научный руководитель Питер Уэйнер предложил в качестве темы будущей диссертации. Требовалось найти способ разбить набор узлов на две группы одинакового размера так, чтобы число ребер, соединяющих узлы из разных групп, было минимальным. Пример такого разбиения показан на рис. 1.7; во всех остальных случаях между группами по пять узлов будет более двух соединений.

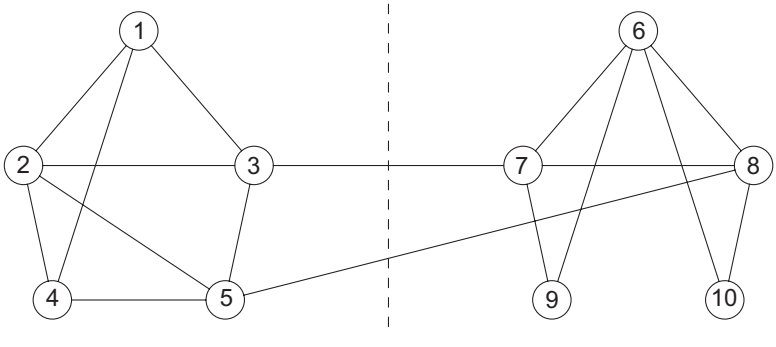


Рис. 1.7. Пример разбиения графов

По официальной версии, в основе этой задачи лежала практическая проблема: как назначить части программы страницам памяти таким образом, чтобы при запуске программы снизить объем подкачки в память и из памяти. Узлы в этом случае соответствуют блокам кода, ребра — возможным переходам из одного блока в другой. Каждое ребро имеет вес, который зависит от частоты переходов и, следовательно, указывает, насколько дорого обходится размещение двух блоков на разных страницах.

В некотором смысле это была искусственно поставленная задача, тем не менее такой абстрактной модели соответствовали и реальные задачи. Например, как разместить на печатных платах компоненты, чтобы снизить расход дорогой проводки, которая соединяет одну с другой? Или менее оправданная задача: как выделить кабинеты сотрудникам, чтобы на одном этаже оказались люди, которые общаются чаще всего?

Для диссертации этого было достаточно, но особых успехов я не добился. Летом 1968 года во время второй стажировки в Bell Labs я описал проблему Шену Лину (рис. 1.8), который недавно разработал наиболее эффективное решение задачи коммивояжера: найти самый выгодный маршрут через все указанные города с возвращением в исходный город.

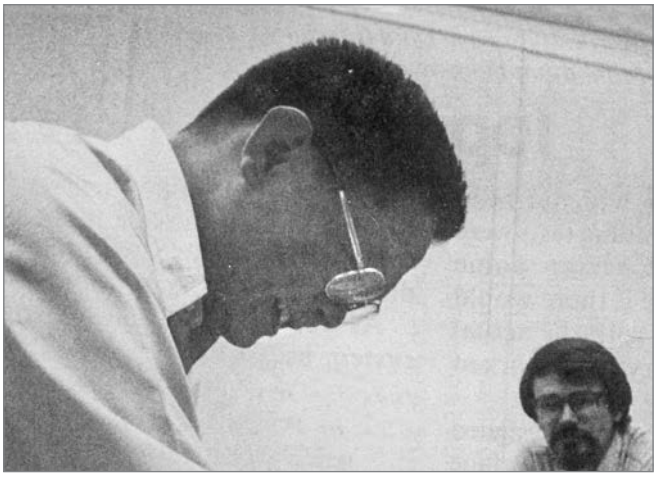


Рис. 1.8. Шен Лин, примерно 1970 год (из архива Bell Labs)

Шен предложил подход к разделению графов, который выглядел многообещающе, хотя и не гарантировал лучшие ответы из возможных. Я понял, как это можно эффективно реализовать, и провел эксперименты на множестве графов, чтобы оценить корректность работы алгоритма. Алгоритм давал нужные результаты, но способа гарантированно получить оптимальное решение мы так и не нашли. Правда, я обнаружил два частных случая, в которых алгоритм работал быстро и давал оптимальные решения. Благодаря этому осенью моя дипломная работа была готова, а в конце января 1969 года я сдал свой последний устный экзамен. Как видите, вместо оптимистично обещанных трех лет я провел в Принстоне четыре с половиной.

Неделю спустя я приступил к работе в Исследовательском центре компьютерных наук. Собеседования я не проходил. Bell Labs прислали мне приглашение еще осенью, оговорив, что сначала я должен защитить диссертацию. Директор центра Сэм Морган сказал, что Bell Labs не берут на работу аспирантов. Так что здорово, что мне удалось окончить аспирантуру. В декабре я получил еще одно письмо, где меня приглашали на более высокую позицию. Получилось, что значительное повышение я получил еще до того, как пришел на работу!

Попутно замечу, что существовала причина, по которой мы не смогли найти алгоритм разбиения графа, всегда дающий оптимальный результат. Просто мы с Шеном в то время о ней не знали. Над задачами комбинаторной оптимизации, такими как разбиение графа, ломали голову и другие ученые, которые обнаружили кое-какие универсальные закономерности. В 1971 году математик и специалист по вычислительной технике из Университета Торонто Стивен Кук показал эквивалентность множества этих сложных задач, включая разбиение графов. Это означало, что эффективный алгоритм (то есть дающий лучшее из возможных решений), найденный для решения одной из них, применим и ко всем остальным. До сих пор полностью не решен вопрос принадлежности таких задач к сложным вычислениям, но я уверен, что они попадают в эту категорию. В 1982 году за эту работу Кук получил премию Тьюринга.

Итак, в 1969 году я стал штатным сотрудником Bell Labs, но не получил никаких указаний, что же мне делать. Это была стандартная практика. Люди знакомились друг с другом, можно было ходить по разным отделам и искать собственную тему исследований. Сейчас такая ситуация может показаться пугающей, но не помню, чтобы я испытывал какие-либо опасения. Вокруг столько всего происходило, что найти тему для исследования было не так сложно, да и после двух лет в роли стажера я уже имел знакомых и был осведомлен о ряде текущих проектов.

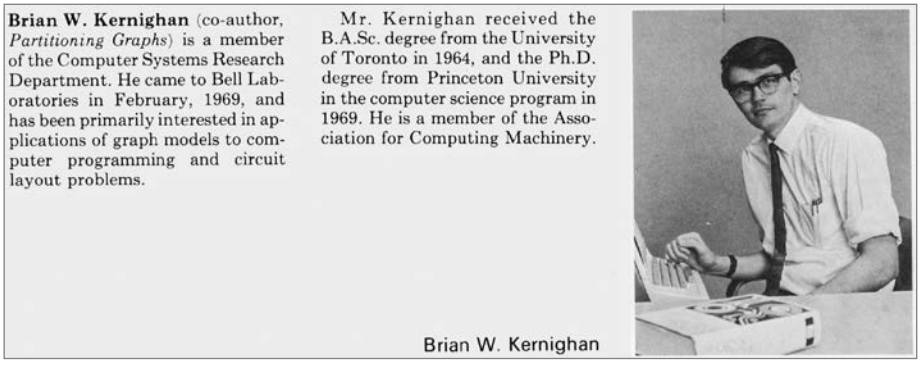
Отсутствие четких указаний было стандартной практикой. Проекты в подразделении 1127 не определялись руководством, а, если так можно выразиться, росли снизу вверх, объединяя людей, которые интересовались определенной темой. По такому же принципу строилась работа с другими подразделениями. В группу разработчиков можно было приглашать коллег-исследователей, и если у них было такое желание, они могли присоединиться.

Некоторое время я продолжал работать с Шеном над комбинаторной оптимизацией. Шен очень хорошо понимал суть таких задач, умел находить подходы к их решению, вручную рассматривая небольшие частные примеры. У него появилась новая идея для задачи коммивояжера, позволяющая значительно улучшить предыдущий алгоритм (уже ставший самым известным). Я реализовал его идею в программе на Фортране. Она прекрасно работала и долгое время считалась достижением.

Это была увлекательная и полезная работа, но, хотя я и умел довольно хорошо преобразовывать идеи в рабочий код, мне так и не удалось разобраться в деталях алгоритмов. Поэтому постепенно я переключился на другие области: программное обеспечение для подготовки документов, специализированные языки программирования и немного письменной работы.

Я пару раз возвращался к сотрудничеству с Шеном, в том числе для работы над сложным инструментом для оптимизации проектирования частных сетей для клиентов AT&T. Мне нравилось переключаться между чистой компьютерной наукой и полезными для компании системами.

Отделу по связям с общественностью очень нравилась работа Шена по решению задачи коммивояжера, поэтому он фигурировал в ряде рекламных публикаций. Размытая фотография со мной в нижнем углу (см. рис. 1.8) взята именно из такой рекламы, а на рис. 1.9 представлена выдержка из какого-то глянцевого рекламного журнала, где рассказывалось о нашей работе по разбиению графов. Скорее всего, эта публикация появилась после того, как мы получили патент на алгоритм.


Brian W. Kernighan (co-author, *Partitioning Graphs*) is a member of the Computer Systems Research Department. He came to Bell Laboratories in February, 1969, and has been primarily interested in applications of graph models to computer programming and circuit layout problems.

Mr. Kernighan received the B.A.Sc. degree from the University of Toronto in 1964, and the Ph.D. degree from Princeton University in the computer science program in 1969. He is a member of the Association for Computing Machinery.

Brian W. Kernighan

Рис. 1.9. Рекламное фото, примерно 1970 год (из архива Bell Labs)

Обратите внимание на галстук. Обычно я их не ношу. Несколько лет спустя мы с Деннисом Ритчи написали статью о языке Си для журнала *Western Electric Engineer*. Журнал попросил прислать наши фотографии, но через несколько недель нам сообщили, что они потеряны и попросили прислать новые, на этот раз в галстуках. Мы наотрез отказались, и вскоре в журнале появилась статья с исходными фотографиями, которые вдруг чудесным образом нашлись.

Постоянную работу я начал в кабинете на пятом этаже здания 2 рядом с лестницей 9. Там я провел 30 лет. Это неподвижная точка в вечно меняющемся мире. Много лет моими соседями были Кен Томпсон, Деннис Ритчи, Боб Моррис, Джо Оссанна и Джерард Хольцманн. Ко мне заходили такие знаменитости, как Джон Лайонс, Энди Таненбаум и Дэвид Уилер.

Последние десять лет моего пребывания в Bell Labs кабинеты Кена Томпсона и Денниса Ритчи находились через коридор от моего. На рис. 1.10 показано фото, которое я сделал в октябре 2005 года, стоя в дверях своего кабинета. Кабинет Кена был левее.

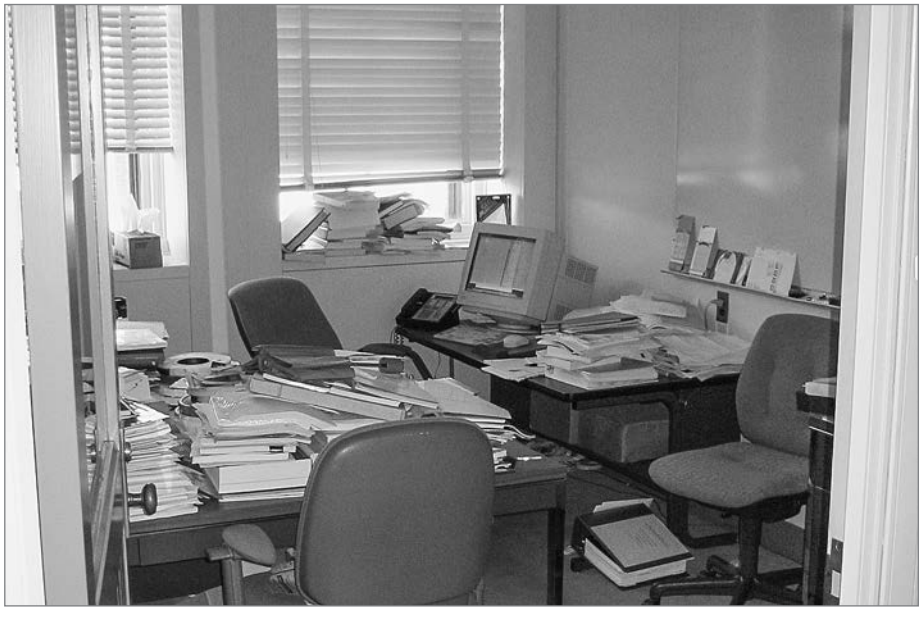


Рис. 1.10. Кабинет Денниса Ритчи в 2005 году

На протяжении многих лет моими ближайшими соседями были Билл Плоджер, Лоринда Черри, Питер Вайнбергер и Аль Ахо. Чуть дальше работали Дуг Макилрой, Роб Пайк и Джон Бентли. Близость местоположения облегчает процесс сотрудничества. С соседями мне очень повезло.

1.5. 137 → 127 → 1127 → 11276

Кем были в то время главные действующие лица? В начале 1970-х годов в Центре компьютерных исследований работало чуть более 30 человек, из которых Unix или вещами, тесно связанными с этой операционной системой, занималось от 4 до 6 сотрудников. На рис. 1.11 представлен фрагмент внутренней телефонной книги Bell Labs. Это не пожелтевшие со временем страницы, штатное расписание изначально было напечатано на желтой бумаге.

Computing Science Research Center

137	Morgan S P, Director, Computing Science Research Center	MH 6490
	Kalainikas Miss E, Secretary	MH 6491
1371	McIlroy M D, Head, Computing Techniques Research Department	MH 6050
	Marky Miss G A, Secretary	MH 6051
	Dimino L A	MH 2390
	Aho A V	MH 4862
	Canaday R H	MH 3038
	Friedman A D	MH 4716
	Jensen P D	MH 6292
	Knowlton K C	MH 2328
	Menon P R	MH 2736
	Morris R	MH 3878
	Neumann P G	MH 2666
	Ossanna J F	MH 3520
	Thompson K L	MH 2394
	Ullman J D	MH 6627
	Wagner Mrs M R	MH 2879
	Weiss Miss R A	MH 2007
1373	Pinson E N, Head, Computer Systems Research Department	MH 2582
	Blejwas Miss V M, Secretary	MH 2583
	Fraser A G	MH 3685
	Johnson S C	MH 3968
	Kernighan B W	MH 6021
	Ritchie D M	MH 3770
	Sturman J N	MH 3164
	Winikoff A W	MH 2661
1374	Brown W S, Head, Computing Mathematics Research Department	MH 4822
	Blejwas Miss V M, Secretary	MH 4823
	Hall A D	MH 4006
	Goldstein A J, Supervisor, Mathematical Techniques Group	MH 2655
	Lin S	MH 2111
	Shafer D M	MH 6862
1374	Traub J F, Supervisor, Numerical Mathematics Group	MH 2383
	Businger P A	MH 2059
	Richman P L	MH 3932
	Schryer N L	MH 2912
1376	Hamming R W, Head, Computing Science Research Department	MH 2064
	Marky Miss G A, Secretary	MH 2065

Рис. 1.11. Телефонная книга Bell Labs, примерно 1969 год (из архива Джерарда Хольцманна)

В 1969 году Исследовательским центром компьютерных наук руководил Сэм Морган (рис. 1.12) — превосходный специалист по прикладной математике и эксперт в теории связи. Сыгравший важную роль в разработке Unix Дуг Макилрой руководил группой, в которую входили Кен Томпсон, Руд Кенедей, Боб Моррис, Питер Ньюманн и Джо Оссанна. В отдел Эллиота Пинсона входили Деннис Ритчи, Сэнди Фрейзер и Стив Джонсон, который тоже много лет был частью команды разработчиков Unix.

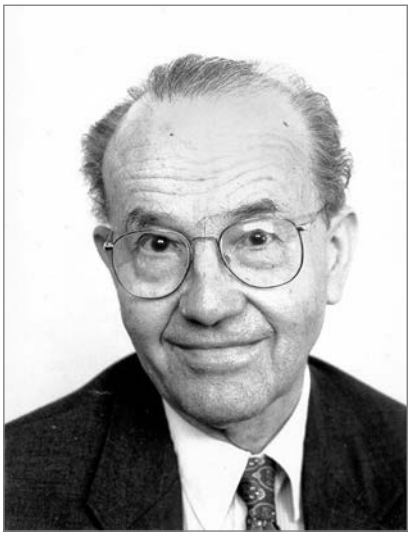


Рис. 1.12. Сэм Морган, руководитель подразделения 1127, примерно 1981 год (из архива Джерарда Хольцманна)

Несмотря на наличие ученых степеней у большинства сотрудников, никто не использовал обращение «доктор»; все обращались друг к другу просто по имени. Было только одно исключение — в телефонной книге перед именами женщин стояло мисс либо миссис, в то время как семейное положение мужчин никак не обозначалось. Я точно не помню, когда эти пометки совсем пропали, но в начале 1980-х их уже не было.

Впрочем, в 1960-х и 1970-х на технических должностях в Bell Labs было мало женщин и представителей других рас; большинство технического персонала составляли белые мужчины, и так продолжалось долгое время. Тогда подобное наблюдалось в большинстве технических сред.

В начале 1970-х Bell Labs запустили три долгосрочные программы, призванные улучшить ситуацию. Программа стипендий для совместных исследований (Cooperative Research Fellowship Program, CRFP) стартовала в 1972 году. Она финансирует четырехлетнее обучение в аспирантуре примерно десяти студентов из числа меньшинств. Аналогичная программа для женщин (Graduate Research Program for Women, GRPW) с 1974 года финансирует обучение 15 или 20 аспиранток в год. В подразделении 1127 в разные периоды работали люди, получившие благодаря этим программам докторскую степень. Большинство из них успешно продолжили карьеру как в Bell Labs, так и в университетах и в других компаниях.

Ежегодно в рамках Летней исследовательской программы (Summer Research Program, SRP), которая также началась в 1974 году, полностью финансируется летняя стажировка примерно 60 студенток и представителей меньшинств, которые размещаются в Мюррей-Хилл, Холмделе, а иногда и в других местах и работают под руководством наставников. Я отвечал за эту программу в подразделении 1127 более 15 лет, познакомился со множеством отличных студентов и несколько раз выступал в роли наставника.

Эти программы оказались эффективными в долгосрочной перспективе, но в 1960-х и 1970-х годах коллектив все еще был достаточно однородным. Впрочем, я не могу утверждать с уверенностью, так как не особо обращал на это внимание.

Иерархия в Bell Labs была очень четкой. На вершине президент, отвечающий за всех сотрудников числом от 15 000 до 25 000 человек. Все направления были пронумерованы: 10 (исследования), 20 (разработка), 50 (телефонная коммутация), 60 (военные системы) и т. п. У каждого направления был свой вице-президент. Исследования проводились в области физики (11), математики и систем связи (13), химии (15) и т. п., каждым из них руководил свой исполнительный директор. Свои исполнительные директора были у юридической и патентной групп. Подразделение математических исследований имело номер 131, а компьютерных наук — 137, причем в последнее входило полдюжины отделов, например 1371. Через несколько лет случилась реорганизация, и мы стали подразделением 127, а со временем появилась дополнительная цифра, превратив наш номер в 1127. Он продержался до 2005 года.

Уровней в нашей иерархии было относительно немного. Такие сотрудники, как я, считались техническим персоналом, и под нами была еще пара технических уровней. У членов технического персонала в отделе исследований обычно был личный кабинет, хотя и предполагалась, что его дверь большую часть времени будет открыта. Над нами предполагался супервизор, хотя за 11 лет в 1127 я помню всего нескольких супервизоров. Дальше шел начальник отдела, в нашем случае Дуг Макилрой, отвечавший за десяток исследователей. Уровнем выше был директор центра, в котором могло быть до десятка департаментов, центрами руководил исполнительный директор, а исполнительных директоров курировал вице-президент.

Вице-президенты были подотчетны президенту. Вице-президентом подразделения исследований с 1955 по 1973 год был выдающийся химик Билл Бейкер. Потом до 1980 года он был президентом Bell Labs. Считалось, что он поименно знает весь технический персонал подотчетного ему направления и то, над чем каждый работает. Думаю, это вполне могло быть правдой. По крайней мере, о нашей с коллегами работе он знал всегда.

До 1981 года я был обычным членом технического персонала, а потом наконец поддался давлению и стал начальником отдела. Большинство неохотно переходило на управляющие должности, потому что это означало уменьшение времени на исследовательскую работу и ответственность за происходящее с другими сотрудниками отдела, что представляло определенные сложности. Но вновь и вновь приходилось слушать обычные аргументы: «Это все равно произойдет, так почему не сделать это сейчас?», «Это может быть ваш последний шанс» или: «Если не вы, то эту должность может занять кто-то менее подходящий».

Как бы то ни было, но я стал руководителем нового отдела 11276 с совершенно бессмысленным названием «Исследование вычислительных структур» (Computing Structures Research). Обычно в отделе было от 8 до 10 сотрудников с широким диапазоном интересов: аппаратное обеспечение машинной графики, средства проектирования интегральных схем, подготовка документов, операционные системы, сети, компиляторы, C++, проектирование беспроводных систем, вычислительная геометрия, теория графов, теория сложности вычислений и многое другое. Было непросто понять, над чем работал каждый из сотрудников, причем понять настолько, чтобы объяснять это вышестоящему руководству. Но это было крайне полезно, и, что удивительно, многое из того, что я тогда узнал, до сих пор живо в моей памяти.

Руководящие должности давали доступ к определенным льготам. Например, более просторному кабинету. Возможно, руководители отделов получали более высокую зарплату, но эта надбавка не была настолько большой, чтобы ее запомнить.

Были и такие привилегии, как ковровое покрытие в кабинетах руководителей, начиная с начальника отдела, в то время как у всех остальных на полу был линолеум или виниловая плитка. После повышения я получил глянцевую брошюру с вариантами ковровых покрытий, офисной мебели и прочего. Я попробовал заказать новый стол, но он оказался слишком большим и неудобным, поэтому я вернулся к старому столу фирмы Steelcase, за которым работал с 1969 года. И отказался от коврового покрытия, так как не был в восторге от подобных иерархических отличий, несмотря на настоятельные рекомендации Сэма Моргана, который утверждал, что подчеркивающая мой авторитет обстановка может пригодиться. Я рад, что в конечном счете все эти знаки различия ушли в прошлое.

Основная задача руководителей отделов состояла в оценке работы сотрудников в течение года в соответствии со сложным ритуалом, который назывался «обзор заслуг». Раз в год все члены технического персонала записывали, что они сделали за год. В подразделении 1127 этот процесс был известен как доклад «Я отличный отчет». Мне кажется, это название придумал Сэм Морган. Начальник отдела писал еще одну бумагу, в которой обобщал и оценивал проделанную работу, в том числе упоминая про «области, требующие улучшения». Этот раздел должен был содержать конструктивную критику.

Написание подобных отчетов и обратная связь дело непростое, а раздел «Области, требующие улучшения» и вообще то и дело норовили оставить незаполненным, пока не пришло распоряжение заполнять его в обязательном порядке. Я придумал фразу «Продолжайте в том же духе», и она выручала меня пару лет, прежде чем мне сказали, что требуется более критический подход, ведь никто не идеален. К счастью, мне не пришлось давать рекомендации по улучшениям такой звезде, как Кен Томпсон. Что я мог ему рекомендовать?

Руководители отделов и директор встречались для согласования оценки деятельности каждого сотрудника. Обычно это занимало целый день. Через несколько недель следовало еще одно занимавшее целый день совещание, на котором определялись зарплаты на следующий год. Хотя эти два меро-

приятия официально назывались оценкой заслуг и пересмотром заработной платы, я всегда считал их «абстрактной оценкой заслуг» и «конкретной оценкой заслуг».

Этот процесс повторялся на более высоком уровне: исполнительный директор рассматривал все результаты работы технического персонала с директорами и оценивал работу руководителей отделов.

В некоторых отделах обзор заслуг мог носить конкурентный характер, но наши обзоры были исключительно товарищескими. Вместо подчеркивания «мои люди лучше, чем ваши» предпочиталось помнить о том, какую пользу принесли нашему отделу другие.

Возможно, я слишком оптимистично смотрю на ситуацию, но мне кажется, что система хорошо работала благодаря полной технической компетентности руководства, которое лично проходило весь процесс на более низких уровнях. Не было сильного уклона ни в сторону практики, ни в сторону теории, по крайней мере у нас в подразделении 1127. Одинаково ценились как хорошие программы, так и хорошие статьи. Положительно сказывалось и отсутствие планов на будущую работу. Ожидалось, что в конце года будет показан некий результат, но не было наказаний за неудачные попытки, а руководство проявляло дальновидность в отношении сотрудников, которые несколько лет работали над одной темой. Мне кажется, помогло еще и небольшое количество управленческих уровней в отделе исследований. Большинство сотрудников не особо заботились о карьерном росте. Тем, кто стремился стать руководителем, больше подходили другие отделы.

В отличие от Bell Labs, в научно-исследовательских вузах на процесс оценки кандидатов при найме или продвижении влияют рекомендательные письма от известных ученых, работающих в той же области. Это приводит к появлению узких специалистов, так как нужно действительно глубоко разбираться в своей области, чтобы рецензенты могли с чистой совестью рекомендовать вас как одного из лучших.

В Bell Labs же ранжирование каждого исследователя осуществлялось снизу вверх. Начальники отделов оценивали своих подчиненных; руководители подразделений в рамках одного центра объединяли эти сведения, затем то же самое происходило на более высоких уровнях, и в конце определялась приблизительная позиция каждого из сотрудников.

Человека, проделавшего большую работу в узкой области, вполне может высоко оценивать его непосредственное руководство, но о его достижениях, скорее всего, никто не будет знать. Междисциплинарная работа ценилась выше, так как была более заметной. Чем шире сотрудничество, тем больше руководителей в курсе дела. В результате появилась организация, в которой всячески поддерживались сотрудничество и междисциплинарные исследования. А так как руководители проходили все стадии процесса, тенденция сохранялась.

Я проработал начальником отдела более 15 лет, был в лучшем случае средним руководителем и оказался только рад уйти в отставку. Другие долгое время успешно сопротивлялись продвижению; Деннис Ритчи стал главой подразделения намного позже, чем я, а Кену Томпсону вообще удалось этого избежать.

Я 20 лет преподавал в университете, но все еще не в восторге от необходимости оценивать работу других людей. Однако без этого не обойтись, а иногда нужно принимать решения, сильно влияющие на жизнь людей, например увольнять (к счастью, мне никогда не приходилось этого делать) или ставить неудовлетворительную оценку на экзамене (нечасто, но так бывает). Аттестация в Bell Labs основывалась на коллективном мнении компетентных специалистов. Как говорил Дуг Макилрой, «гениальность системы состояла в коллегиальности. Никто не зависел от отношений с одним начальником». Процедура служебной аттестации в Bell Labs была не идеальной, но весьма неплохой, я слышал и читал про намного худшие варианты.

ГЛАВА 2

ЗАРОЖДЕНИЕ UNIX (1969)

> И вдруг я понял, что до создания операционной системы осталось три недели.
>
> *Кен Томпсон, на компьютерной выставке*
> *Vintage Computer Festival East, 4 мая 2019 года*

Операционная система Unix родилась в 1969 году, но это произошло не на пустом месте. Потребовался опыт нескольких сотрудников Bell Labs, которые работали над другими операционными системами и языками. Сейчас я расскажу вам, как это случилось.

2.1. НЕБОЛЬШАЯ ТЕХНИЧЕСКАЯ СПРАВКА

Этот раздел представляет собой краткое введение в основной материал: компьютеры, аппаратное и программное обеспечение, операционные системы, программирование и языки программирования. Те, кому все это уже знакомо, могут пропустить эту информацию, остальным же, я надеюсь, она поможет быстро понять, о чем пойдет речь. Более подробные объяснения можно найти в моей книге *Understanding the Digital World*.

Компьютер, по сути, не более чем калькулятор, просто выполняющий вычисления очень быстро. Современные компьютеры делают миллиарды вычислений в секунду, а в 1970-х они работали значительно медленнее.

В репертуаре среднестатистического компьютера в 1960-х и 1970-х годах было нескольких десятков инструкций. Он умел выполнять арифметические действия (сложение, вычитание, умножение, деление), считывал информацию из оперативной памяти и сохранял туда данные, взаимодействовал с такими устройствами, как диски и все с ними связанное. Кроме того, были инструкции, которые, базируясь на предыдущих действиях компьютера, указывали, что он будет делать дальше. Таким способом компьютер сам определяет свои следующие действия.

Инструкции и данные хранятся в той же оперативной памяти, которая обычно называется запоминающим устройством с произвольным доступом (random access memory, RAM). Действия компьютера зависят от загруженного в эту память набора инструкций. Когда пользователь нажимает иконку программы Word или браузера Chrome, операционная система получает команду загрузить в память инструкции для соответствующей программы и начать ее выполнение.

Программированием называется процесс создания последовательностей операций, выполняющих некоторую задачу. Для этого используется какой-то язык программирования. Инструкции можно задавать и напрямую, но даже в случае крошечных программ это сложная рутинная работа со множеством деталей, поэтому и создавались языки программирования, позволяющие людям формулировать вычислительные задачи более понятным для них способом. Программы, называемые *компиляторами* (которые тоже кто-то должен был написать), переводят с удобных человеку языков высокого уровня на язык отдельных инструкций, понятный компьютеру конкретного типа.

Операционная система — это тоже программа, просто большая и сложная. Но она состоит из тех же инструкций, что и обычные программы, такие как текстовый процессор Word или браузер. Ее задача — контролировать все запускаемые программы и управлять взаимодействием с собственно компьютером.

Для наглядности рассмотрим небольшой пример. Предположим, мы знаем длину (length) и ширину (width) прямоугольника и нужно вычислить его площадь (area). По-русски мы скажем, что «площадь — это произведение длины и ширины». На доске в школе учитель напишет это в виде формулы

$$площадь = длина \times ширина$$

На языке программирования высокого уровня эта формула будет выглядеть так:

```
area = length * width
```

Причем это верно для большинства современных популярных языков. Компилятор превращает такую запись во все еще читабельную, но уже машинно-ориентированную последовательность инструкций. Для некоего среднего компьютера эта последовательность может выглядеть так:

```
load length
multiply width
store area
```

Наконец, программа, называемая *ассемблером*, преобразует эту последовательность более или менее читабельных инструкций в последовательность машинных инструкций, которые можно загрузить в оперативную память компьютера, где они будут выполнены. Тут опущено множество деталей, например как мы определяем компиляцию и загрузку, каким образом передаем компьютеру информацию о длине и ширине, как выводится информация о площади и многое другое, но суть происходящего я изложил.

Если вы предпочитаете работающие примеры, вот программа на языке Си, которая считывает данные о длине и ширине и выводит площадь прямоугольника:

```
void main() {
    float length, width, area;
    scanf("%f %f", &length, &width);
    area = length * width;
    printf("площадь = %f\n", area);
}
```

Эту программу можно скомпилировать и выполнить на любом компьютере.

Названия современных операционных систем Windows и macOS слышал, наверное, каждый; смартфоны управляются такими ОС, как Android и iOS.

Операционной системой называется программа, управляющая компьютером, которая распределяет ресурсы между работающими программами. Она отвечает за выделение этим программам нужного количества оперативной памяти. Именно операционная система позволяет на ПК или ноутбуке

одновременно запустить браузер, текстовый процессор, музыкальный плеер, а возможно, еще и небольшую программу для вычисления площади прямоугольника. По мере необходимости она переключает свое внимание с одной программы на другую.

Она же по запросу отображает на экране конкретную программу и управляет устройствами хранения, такими как жесткие диски, позволяя сохранить документ Word, чтобы позже его можно было открыть и продолжить работу.

Еще ОС координирует взаимодействие с интернетом, позволяя браузеру одновременно выполнять поиск, общаться с друзьями, делать покупки и выкладывать видео с котиками.

Людям, далеким от программирования, это не так очевидно, однако операционная система защищает программы от других программ при возникновении ошибок, а также саму себя от некорректно работающих или вредоносных программ и пользователей.

Аналогичным образом обстоят дела с операционными системами на телефонах. Они выполняют множество действий в фоновом режиме, поддерживая связь через мобильную или беспроводную сеть. В основе телефонных приложений лежит та же идея, что и в основе таких программ, как Word. Они отличаются деталями реализации, но написаны на одних и тех же языках.

Современные операционные системы — большие и сложные программы. В 1960-х годах все было проще, но для своего времени они тоже были большими и сложными. Обычно производители компьютеров, например IBM или DEC (Digital Equipment Corporation), предлагали для своего оборудования одну или несколько операционных систем. Между оборудованием от разных производителей и иногда даже между предложениями одного производителя не было ничего общего, соответственно, сильно отличались друг от друга и операционные системы.

Еще больше усложняло ситуацию то, что операционные системы писались на языке ассемблера. Этот язык является понятным человеку представлением машинных инструкций с привязкой к конкретному типу аппаратного обеспечения. Для каждой аппаратной платформы был свой язык ассемблера.

Отсутствие общности между системами, требовавшее использования несовместимых друг с другом низкоуровневых языков, сильно затрудняло прогресс. Ведь при переходе на другую операционную систему или архитектуру все программы нужно было фактически переписывать с нуля. Вскоре вы увидите, что проект Unix породил операционную систему, подходящую для всех видов аппаратного обеспечения, причем написанную не на ассемблере, а на языке высокого уровня, что позволяло сравнительно легко переносить ее с одной платформы на другую.

2.2. CTSS И MULTICS

Самой инновационной операционной системой того времени была CTSS (Compatible Time-Sharing System, совместимая система разделения времени), созданная в MIT в 1964 году. Большинство операционных систем того времени обрабатывали данные в «пакетном режиме». Программы записывались на перфокарты (это было очень давно!), которые передавали оператору, а затем ждали результатов часами и даже днями.

Перфокарты делались из тонкого картона и вмещали до 80 символов. Как правило, перфокарта содержала одну строку программы, соответственно, для нашей программы на языке Си потребовалось бы шесть карт. Изменения в программу вносились путем замены перфокарт. На рис. 2.1 показана стандартная карта на 12 строк и 80 колонок.

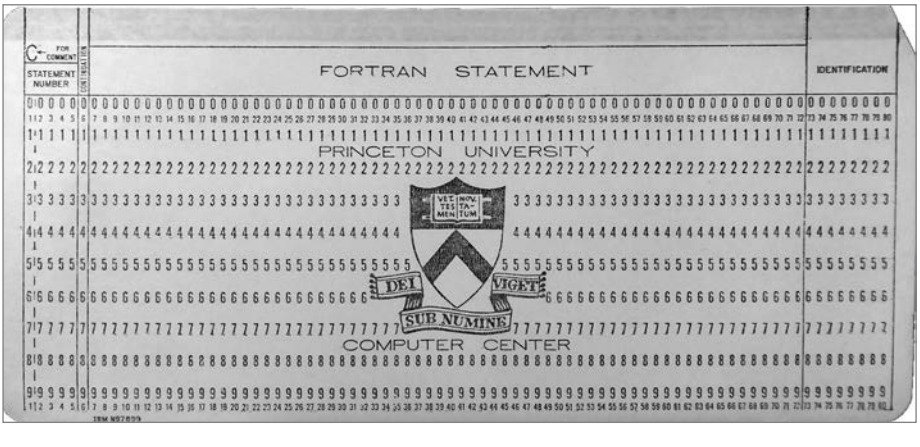


Рис. 2.1. Перфокарта 7-3/8 на 3-1/4 дюйма (187,325 на 82,55 мм)

Программисты, работавшие в операционной системе CTSS, использовали устройства, похожие на пишущие машинки («терминалы», напоминающие телетайпы Model 33, пример которого вы увидите на рис. 3.1 в следующей главе). Эти устройства напрямую или через телефонную линию подключались к большому компьютеру, IBM 7094, память которого вдвое превышала обычные 32 Кбайт слов. Операционная система умела быстро переключаться между всеми активными пользователями, создавая у каждого иллюзию, что весь компьютер в его распоряжении. Эта функциональность называлась «разделение времени» и (насколько я помню) была намного приятнее и продуктивнее пакетной обработки. Часто мне действительно казалось, что других пользователей у этого компьютера нет.

Операционная система CTSS оказалась настолько продуктивной средой программирования, что в MIT решили создать ее усовершенствованную версию, которая могла бы предоставлять вычислительные услуги большому числу пользователей. Поэтому в 1965 году началось проектирование системы Multics (сокращение от Multiplexed Information and Computing Service — Мультиплексная информационная и вычислительная служба).

Проект Multics требовал больших трудозатрат, ведь он был связан с созданием нового перспективного программного обеспечения и с аппаратным обеспечением, возможности которого превосходили IBM 7094, поэтому MIT попросил помощи у сторонних организаций. Компания General Electric, в то время производившая компьютеры, должна была спроектировать и создать компьютер с новыми аппаратными функциями для лучшей поддержки разделения времени и многопользовательского режима. К разработке операционной системы привлекли Bell Labs, у которых с начала 1950-х был большой опыт создания собственных систем.

Задача стояла крайне сложная, и вскоре возникли проблемы. Можно сказать, что разработчики столкнулись с *эффектом второй системы*. Этот термин означает ситуацию, когда на волне успеха первой системы (например, CTSS) начинаются попытки не только исправить все ее недочеты, но и добавить всю ту функциональность, которая в нее не вошла. Как следствие одновременного принятия множества решений появляется переусложненная система. Именно так и получилось с Multics. Словосочетание «избыточно сложная» встречается в разных описаниях этой системы, а Сэм Морган назвал ее «попыткой забраться на множество деревьев одновременно». Кроме того, не нужно быть семи пядей во лбу, чтобы предсказать

сложности при реализации проекта, над которым работают две компании и университет, расположенные в разных местах.

С 1966 по 1969 год над операционной системой Multics работали такие исследователи из Bell Labs, как Дуг Макилрой, Деннис Ритчи, Кен Томпсон и Петер Нойман, который заменил Вика Высотского после перевода последнего в другой отдел Bell Labs. Дуг хорошо знал «язык программирования номер один» (PL/I), который должен был использоваться для написания Multics. Деннис, который в то время был студентом Гарварда, работал над документацией Multics и над подсистемой ввода-вывода. Той же подсистемой занимался Кен. Полученный при этом опыт пригодился ему позднее при разработке Unix, хотя в интервью 2019 года он описывал свою работу над Multics как «производство того, чем я сам пользоваться не хотел».

К 1968 году стало ясно, что, хотя Multics — хорошее программное обеспечение для небольшой группы людей, цель получить систему, предоставляющую недорогие вычислительные услуги для Bell Labs, не достигнута. Система оказалась слишком дорогой. Поэтому в апреле 1960 года Bell Labs вышли из проекта.

Впрочем, проект Multics в конечном итоге завершили. По крайней мере, было объявлено об его успешном завершении. Система поддерживалась и использовалась до 2000 года, хотя и не получила широкого распространения. Она породила множество хороших идей, но самый весомый ее вклад оказался неожиданным: то, как она повлияла на скромную операционную систему Unix, разработка которой отчасти явилась реакцией на сложность Multics.

2.3. ЗАРОЖДЕНИЕ UNIX

После выхода из проекта Multics сотрудникам Bell Labs, которые работали над этой операционной системой, пришлось искать себе новое занятие. Кен Томпсон (рис. 2.2) хотел продолжить разработку операционных систем, но руководство Bell Labs, разочарованное результатом с Multics, не было заинтересовано в покупке оборудования под аналогичный проект. В результате Кен и его коллеги занимались проектированием различных компонентов операционной системы на бумаге, никак не реализуя это на практике.

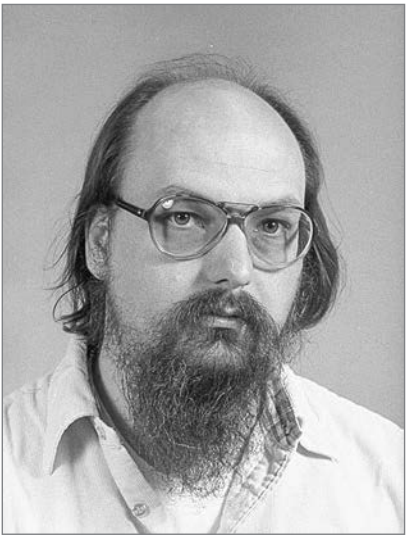


Рис. 2.2. Кен Томпсон, примерно 1981 год (из архива Джерарда Хольцманна)

Через некоторое время Кен обнаружил малоиспользуемый компьютер PDP-7 от DEC, в основном служивший устройством ввода для проектируемых электронных схем.

Первые поставки PDP-7 начались в 1964 году, но компьютеры развивались настолько быстро, что к 1969 году он уже считался устаревшим. Машина была не очень мощной, с памятью на всего 8 Кбайт 18-битных слов (16 Кбайт), но с хорошим видеопроцессором, поэтому Кен написал для нее игру Space Travel. Игрок мог путешествовать по Солнечной системе и приземляться на разные планеты. Это было очень увлекательно, и я мог играть часами.

У PDP-7 была интересная периферия — очень высокий 8-дюймовый дисковод. У этих приводов была довольно мощная пружина для выброса диска, и поэтому массивная дискета при открывании защелки могла чувствительно ударить по рукам. Считалось, что если что-то сломается, то перед машиной лучше не стоять. Дисковая подсистема для такого компьютера была слишком быстрой. Для устранения этого недостатка Кен написал алгоритм планирования работы диска, который пытался до максимума увеличить его пропускную способность.

Оставался вопрос, где его протестировать. Нужно было загрузить на диск данные, и Кен решил написать программу, которая сделает это.

«В какой-то момент я понял, что от операционной системы меня отделяют три недели. Требовалось написать три программы, по одной в неделю: редактор, в котором будет создаваться код; ассемблер для трансляции этого кода на язык, понятный компьютеру PDP-7; и оверлей ядра, который можно назвать операционной системой».

Как раз в то время жена повезла их годовалого сына к родителям Кена в Калифорнию, так что он мог спокойно работать. В интервью 2019 года он описал это так: «Неделя, неделя и еще неделя — и у нас появилась Unix». По всем параметрам это был очень продуктивный результат.

Спустя пару лет после нашего с Кеном ухода из Bell Labs я попросил его вспомнить, как он за три недели написал первую версию Unix. Вот его дословный ответ:

```
Дата: Четверг, 9 января 2003 13:51:56 -0800
Я писал файловую систему для проверки пропускной способности и прочего
и столкнулся с трудностями при загрузке данных. Я мог поместить в цикл
вызовы read и write, но что-либо более сложное было практически недоступно.
Как раз тогда Бонни отправилась навестить моих родителей в Сан-Диего.
Я решил, что все это почти напоминает систему с разделением времени,
не хватает только вызова exec, оболочки, редактора и ассемблера
(компиляторов не предполагалось). Добавить вызов exec было просто, остальные
3 задачи отняли у меня по неделе каждая, как раз столько времени не было
Бонни.
Машина была 8 Кбайт 18-битных слов, из которых 4 Кбайт ушло под ядро, а 4 Кбайт
выделялось меняющимся пользователям.
Кен
```

Эта первая версия знакомой всем системы Unix функционировала с середины до конца 1969 года, поэтому можно сказать, что именно так произошло ее рождение.

Работала в ней небольшая группа пользователей: разумеется, Кен и Деннис, кроме того, Дуг Макилрой, Боб Моррис, Джо Оссанна и, благодаря некоторому везению, я. У каждого из нас был числовой идентификатор пользователя. Идентификаторы существовали не только у людей, но и у системных функций. Суперпользователь имел идентификатор 0. Был

и ряд других особых случаев. Идентификаторы реальных пользователей начинались с 4. Мне кажется, у Денниса был 5, у Кена — 6, а у меня — 9. Одноразрядный идентификатор пользователя в первой версии Unix можно считать знаком отличия.

2.4. ПРОИСХОЖДЕНИЕ НАЗВАНИЯ

В какой-то момент написанная для компьютера операционная система PDP-7 приобрела имя, хотя я уже не могу точно вспомнить, как это было.

Помню, как, стоя в дверях своего кабинета, я разговаривал с коллегами, среди которых были Кен, Деннис и Петер Нойманн, и предложил, основываясь на латинских корнях, заменить multi на uni, ведь система Multics предлагала «много всего», а в новой системе есть самое большее по одному из разного.

Есть и альтернативное воспоминание, согласно которому Петер Нойманн придумал название UNICS как аббревиатуру от UNiplexed Information and Computing Service. Вот как говорит он сам:

> Хорошо помню, как однажды утром во время ланча Кен рассказал, что накануне написал ядро однопользовательской ОС на тысячу строк для компьютера PDP-7, одолженного ему Максом Мэтьюсом. Я предложил ему превратить эту систему в многопользовательскую, и на следующий день была написана еще тысяча строк. Именно так появилась концепция UNICS как «кастрированной версии Multics».

Подробностей Петер припомнить не смог, так что не знаю, насколько заслуженно, но я могу приписать изобретение названия себе.

Впрочем, аббревиатура UNICS каким-то образом превратилась в Unix, что выглядело намного лучше. (Ходили слухи, что юристам AT&T не понравилось сходство в произношении слов Unics и eunuch, то есть евнух.) Впоследствии Деннис Ричи характеризовал название как «предательское обыгрывание» названия Multics, что на самом деле и произошло.

2.5. БИОГРАФИЯ: КЕН ТОМПСОН

В мае 2019 года на выставке Vintage Computer Festival East в городе Уолл, штат Нью-Джерси, я взял у Кена небольшое интервью. Задавал ему наводящие вопросы и слушал ответы. Именно этот материал, который вы можете найти на YouTube[1], послужил основой данного раздела.

Кен родился в 1943 году. Его отец был военным моряком, и в детстве Кену пришлось попутешествовать. Он жил в разных местах, в том числе в Калифорнии и Луизиане, и даже провел несколько лет в городе Нейплсе штата Флорида.

Он поступил в Калифорнийский университет в Беркли, чтобы изучать электротехнику — эта тема интересовала его с детства. Учиться было легко, так как еще до поступления он десять лет увлекался радиоэлектроникой. Кроме того, в Беркли он открыл для себя информатику.

Я помешался на компьютерах, я их обожал. В то время учебной программы по компьютерным наукам в Беркли не было, ее только разрабатывали.

Летом после выпуска я не знал, куда себя деть. Окончание учебы оказалось для меня сюрпризом, потому что я не чувствовал себя компетентным специалистом.

Я просто собирался остаться в университете, потому что… это был мой университет. Я был по уши в нем. Огромный главный компьютер отключали в полночь, но у меня были свои ключи, я приходил, включал его, и до 8 утра это был мой персональный компьютер.

Я был счастлив. У меня не было амбиций. Я был трудоголиком, но без цели.

В последний год обучения Кен в качестве вольнослушателя посещал курс профессора Элвина Берлекэмпа, который вскоре перешел в Bell Labs. После выпуска Кен не стал подавать заявление в магистратуру, потому что ему казалось, что он недостаточно хорош для этого.

В конце лета Берлекэмп сказал: «Вот твое расписание в магистратуре». Оказалось, что он подал заявление вместо меня, и меня приняли!

---

[1] https://www.youtube.com/watch?v=EY6q5dv_B-o

В 1966 году после получения степени магистра Кен получил приглашения от нескольких компаний, в том числе от Bell Labs, но заявил, что не хочет работать в крупных компаниях.

Впрочем, специалист по подбору кадров не оставлял свои попытки. По словам Кена, он отверг шесть или восемь попыток найма из Bell Labs, и все из-за отсутствия амбиций. Он помнит, как к нему в дверь постучался агент по найму из Bell Labs, а Кен предложил ему имбирное печенье и пиво. (Наверное, это была какая-то странная калифорнийская диета.)

Все кончилось тем, что Кен согласился поехать в Нью-Джерси за счет Bell Labs, но всего на день, чтобы навестить университетских друзей. Но когда он там очутился, его поразили имена, которые он увидел:

> Я шел по коридору Исследовательского центра компьютерных наук и читал имена на дверях кабинетов, и все они были знакомыми. Это было потрясающе. Собеседование со мной проводили два удивительных человека... одним из них был Шен Лин.
>
> Я не знаю, каким образом они отследили меня, но когда в арендованном автомобиле ехал в сторону Восточного побережья, во время третьей остановки я обнаружил ожидающее меня предложение. Я поехал назад и пару часов размышлял об этом. А потом во время остановки в доме еще одного моего друга позвонил им и сказал, что согласен.

Итак, Кен стал сотрудником Bell Labs в 1966 году и приступил к работе над операционной системой Multics, которая затем, как я уже описывал выше, перешла в работу над Unix.

Кен с детства увлекался играми, в частности шахматами. Проигрывать он не любил, а выиграв, расстраивался из-за проигрыша соперника, поэтому предпочитал роль зрителя. В 1971 году он написал шахматную программу для компьютера PDP-11, после чего началась работа над аппаратным обеспечением для ускорения вычислений, например определения возможных перемещений фигуры из заданной позиции. В конечном итоге это привело к появлению компьютера для игры в шахматы, который назвался Belle (рис. 2.3). Разработку Belle Кен и Джо Кондон вели с 1976-го по начало 1980-х.

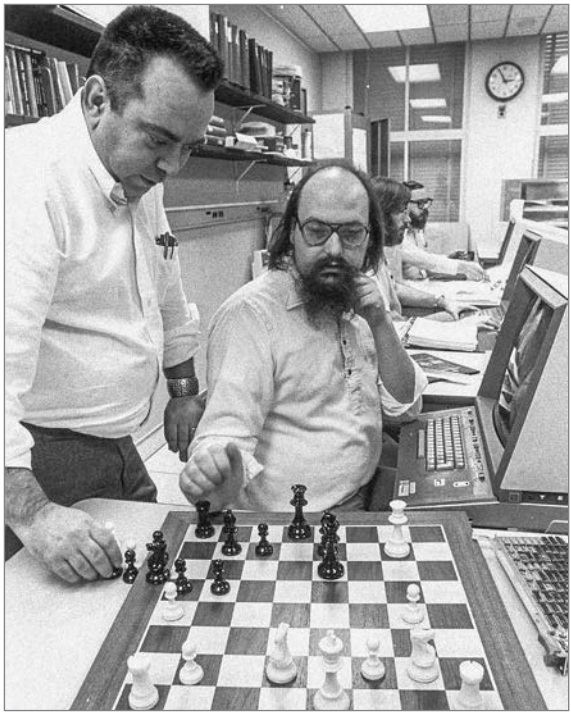


Рис. 2.3. Кен Томпсон и Джозеф Кондон (Музей компьютерной истории)

Компьютер Belle (рис. 2.4) стал первой машиной, достигшей уровня мастера по шахматам. Его рейтинг Эло составил 2200[1], а в 1980 году он выиграл чемпионат мира по шахматам среди компьютерных программ и несколько североамериканских чемпионатов, а затем был отправлен на покой в Смитсоновский институт.

Для Международной ассоциации компьютерных шахмат Деннис Ритчи написал статью о деятельности Кена Томпсона. Ее можно найти по адресу www.bell-labs.com/usr/dmr/www/ken-games.html. Там рассказывается, что интерес Кена к играм не ограничивался только шахматами. Добавлена в статью и шахматная партия, в ходе которой Belle выиграл у программы Blitz 6.5 на Североамериканском чемпионате 5 декабря 1978 года, с ком-

[1] Метод расчета относительной силы игроков в играх, в которых участвуют двое игроков, систему разработал американский профессор физики венгерского происхождения Арпад Эло. Шахматный рейтинг 2200 соответствует уровню национального мастера. — *Примеч. ред.*

ментариями пионера компьютерных шахмат Монти Ньюборна и международного мастера Дэвида Леви:

> 1. e4 e5 2. Nf3 Nc6 3. Nc3 Nf6 4. Bb5 Nd4 5. Bc4 Bc5 6. Nxe5 Qe7 7. Bxf7+ Kf8 8. Ng6+ hxg6 9. Bc4 Nxe4 10. O-O Rxh2!! 11. Kxh2 {ускорение потерь} Qh4+ 12. Kg1 Ng3 13. Qh5 {попытка задержки} gxh5 14. fxg3+ Nf3# {возможно, уникальный уход от шаха, двойной шах и одновременный мат; "самая красивая комбинация, созданная современной компьютерной программой... мы стали свидетелями начала новой эры в компьютерных шахматах"}.

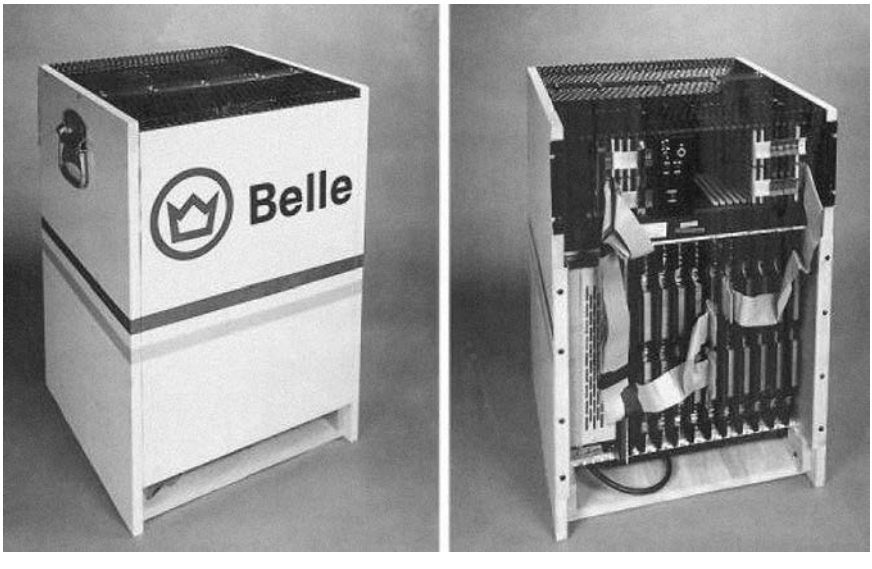


Рис. 2.4. Шахматный компьютер Belle (фото предоставлено Музеем компьютерной истории)

Шахматная партия может закончиться победой, поражением или ничьей. Согласно правилу 50 ходов, игрок имеет право потребовать ничью, если на протяжении последних 50 ходов каждого игрока ни одна фигура не была взята и ни одна пешка не сделала хода. Это правило не позволяет затягивать игру до бесконечности, рассчитывая, что соперник устанет и сделает ошибку.

Кен решил определить, насколько корректно выбрано число 50. С помощью Belle и сложной системы баз данных он оценил все четырех- и пятифигурные окончания игр и обнаружил, что некоторые из них можно выиграть бо-

лее чем за 50 ходов. К этому времени Кена уже хорошо знали в мире шахмат, и иногда в Bell Labs появлялись гроссмейстеры, желающие попробовать свои силы в поединке с Belle, особенно в эндшпиле. Так что мне доводилось встречаться с чемпионами мира Анатолием Карповым и Виши Анандом.

Еще Кен был заядлым пилотом и регулярно летал по Нью-Джерси как сам, так и с гостями, вылетая из аэропорта в Морристауне. Он заинтересовал полетами других членов подразделения 1127, и в какой-то момент среди сотрудников оказалось полдесятка частных пилотов. Обычно они все вместе летали посмотреть на листопад или пообедать в каком-нибудь интересном месте. Дуг Макилрой вспоминает:

> Благодаря любви Кена к пилотированию и телескопам Роба Пайка мы наблюдали не только листопад в Новой Англии, но и затмение в горах Адирондак. Летали мы и наблюдать прохождение Меркурия по диску Солнца. Астрономическая тема в команде Unix началась с программы `azel` Джо Оссанны, предназначенной для управления наземной станцией спутников Telstar. С ее помощью мы обычно определяли, где найти искусственные спутники. Затем появилась программа `sky` Боба Морриса и программа, прогнозирующая наступление различных небесных явлений, написанная Кеном, звездные карты Ли Макмэхона, составленные с помощью моей программы `map`, и, наконец, программа Роба `scat` для поиска каталогов звездного неба в заданной области.

В декабре 1992 года Кен и Фред Грамп отправились в Москву, чтобы вместо привычных легких самолетов Cessnas полетать на истребителе МиГ-29. Фотографии 2.5 и 2.6 показывают Кена, готового к взлету и рулящего самолетом после посадки.

Кен, как и я, уволился из Bell Labs в конце 2000 года. Я стал работать в Принстоне, а он присоединился к стартапу Entrisphere, который основали наши бывшие коллеги. В 2006 году он перешел в Google, где вместе с Робом Пайком и Робертом Гризмером создал язык программирования Go. Когда до меня дошли слухи о его переходе в Google, я попросил у него подтверждения. И вот что он ответил:

```
Дата: среда, 1 ноября 2006 16:08:31 -0800
Тема: Re: голос из прошлого
Это правда. На медианный возраст сотрудников Google это сильно не повлияло,
а вот средний возраст сильно изменился.
Кен
```

Рис. 2.5. Кен Томпсон, готовящийся к взлету (фото предоставлено сайтом cat-v.org)

Рис. 2.6. Самолет Кена на взлетной полосе после посадки (фото предоставлено сайтом cat-v.org)

ГЛАВА 3

ПЕРВАЯ РЕДАКЦИЯ (1971)

Это руководство содержит полное описание всех общедоступных функций Unix. Здесь вы не найдете ни общего обзора (см. «Unix — система с разделением времени»), ни подробностей внедрения системы (которые еще нужно будет раскрыть).

Первая редакция руководства для программистов Unix,
3 ноября 1971 года

ОШИБКИ: команда `rm` должна спрашивать, действительно ли требуется удалить файл, предназначенный только для чтения.

Раздел руководства, посвященный команде `rm`*,*
3 ноября 1971 года

Созданная для компьютера PDP-7 операционная система оказалась достаточно интересной, чтобы, даже несмотря на малую мощность компьютера и недостаток программного обеспечения, ею начали пользоваться. С точки зрения небольшой группы сотрудников, эта вычислительная среда была более продуктивной, чем большой центральный компьютер. В результате Кен Томпсон, Деннис Ритчи и остальные сделали попытку получить новую машину, способную поддерживать больше пользователей и проводить более интересные исследования.

Первым делом они предложили приобрести компьютер PDP-10 от фирмы DEC, популярный в то время в университетах и других исследовательских лабораториях. Он напоминал IBM 7090 с 36-битными словами, а также GE 635 и 645 и намного превосходил мощностью маленький PDP-7. Но на такую покупку требовалось полмиллиона долларов.

При этом воспоминания о неудачном опыте с Multics были еще слишком живы, и подобное предложение не имело шансов на успех. По словам Кена, руководство декларировало: «Мы не занимаемся операционными системами», что вполне можно было трактовать как «мы не собираемся выделять деньги на более мощную машину».

С моей точки зрения, подобная осторожность начальства оказывает позитивное влияние, ведь в этом случае для получения ресурса людям приходится продумывать свои предложения и концентрироваться на продвигаемой идее. Ограниченность ресурсов зачастую заставляет идеально планировать рабочий процесс. Как бы то ни было, группа Unix выдвинула другую идею — приобрести новый мини-компьютер PDP-11, который только что начала продавать фирма DEC. Он стоил уже не полмиллиона, а всего 65 тысяч.

Это предложение тоже успеха не имело. В интервью Майку Махони 1989 года Сэм Морган так пояснил причины отказа:

> Наши принципы управления заключались в том, что мы нанимали талантливых людей, знакомили их с условиями работы, в общих чертах обрисовывали, что мы хотим от них получить, и предоставляли полную свободу. Это не означало обязательного финансирования всех идей. Руководство проявляло избирательный подход по поводу того, что делали сотрудники. Разумеется, есть риск ошибочно отвергнуть хороший проект, но опыт показывает, что сильная идея все равно пробьет себе дорогу.

Оглядываясь назад, могу сказать, что даже хорошо, что нас заставляли работать в условиях ограничений. Как сказал в своей речи на вручении премии Тьюринга 1983 года Кен:

> Популярность Unix стала следствием общеотраслевого перехода с центральных универсальных ЭВМ на автономные мини-компьютеры. Я подозреваю, что если бы Даниэлю Боброву из-за отсутствия возможности приобрести PDP-10 пришлось согласиться на PDP-11, сейчас вместо меня тут бы стоял он.

Даниэль Бобров был одним из основных разработчиков операционной системы Tenex, написанной в 1969 году для PDP-10.

3.1. UNIX ДЛЯ ЗАЯВОК НА ПАТЕНТЫ

Прямые просьбы приобрести новое оборудование успеха не имели, но был альтернативный путь. Bell Labs как крупная и продуктивная научно-исследовательская компания генерировали множество патентных заявок. Примерно по патенту в день. Такая заявка представляла собой текстовый документ с жесткими требованиями к формату, например нумерации строк на странице. Ни одна существующая система не умела оформлять документы таким способом, поэтому патентный отдел планировал закупить оборудование у компании, обещавшей программное обеспечение, которое со временем позволит решать подобные задачи. В тот момент нумерация строк еще не была реализована.

Джо Оссанна предложил другой план. Патентный отдел сможет пользоваться для подготовки патентных заявок компьютером PDP-11, потому что группа Unix напишет программное обеспечение для вывода заявок в нужном формате. Про работу над операционными системами речи не шло.

Этот комбинированный вариант позволил обойти остаточные возражения руководства. Деньги на PDP-11 пришли от директора Центра речевых и акустических исследований (Speech and Acoustics Research Center) Макса Мэтьюса. Дело в том, что один из его начальников, Ли Макмэхон, очень интересовался обработкой текста и содействовал продвижению плана Оссанны.

Сделка была одобрена, PDP-11 куплен, а Кен и Деннис быстро преобразовали версию Unix для PDP-7 под новую машину. Ресурсы нового компьютера тоже были ограничены: всего 24 Кбайт оперативной памяти и полмегабайта дискового пространства. На нужды операционной системы выделили 16 Кбайт, и еще 8 осталось под пользовательские программы.

Джо Оссанна написал программу Nroff («new roff»), сходную с системой подготовки текстов Roff, которая могла выводить патентные заявки в требуемом формате. Во второй половине 1971 года работа над патентными заявками осуществлялась исключительно в Unix. Подробно о форматировании текстов я расскажу в главе 5.

Этим занимались днем. А ночами Кен, Деннис и другие на том же PDP-11 разрабатывали программное обеспечение. Разработку приходилось вести ночью, чтобы не мешать машинисткам. Кроме того, следовало соблюдать осторожность. У PDP-11 не было механизмов аппаратной защиты, не позволяющих программам влиять друг на друга или на операционную систему, поэтому ошибка могла легко привести к сбою системы, а ошибка в файловой системе — к потере всех достигнутых результатов. Впрочем, опыт оказался настолько успешным, что патентный отдел приобрел еще один компьютер PDP-11 специально для группы Unix, после чего разработка стала вестись в течение рабочего дня. Появившаяся версия стала первой редакцией Unix.

Фотография 3.1 была сделана отделом связей с общественностью в 1972 году. На ней Кен Томпсон и Деннис Ритчи работают на компьютере PDP-11 с ранней версией операционной системы Unix. Это компьютер модели

Рис. 3.1. Кен (сидит) и Деннис у компьютера PDP-11, примерно 1972 год (Википедия)

PDP-11/20. Маленькие круглые штуки на уровне головы — носители информации на магнитной ленте DECtapes, хранящие 144 K 18-битных слов. Они допускали чтение и запись отдельных блоков данных и играли роль временных дисков, медленных, но надежных. Ленты были съемными, поэтому применялись и для резервного копирования.

Кен печатал на телетайпе модели 33, прочном, но медленном и шумном устройстве. Фактически это была управляемая компьютером электромеханическая пишущая машинка, позволяющая печатать только в верхнем регистре со скоростью 10 символов в секунду. Модель 33 появилась в 1963 году, но более ранние версии широко использовались с начала 1930-х.

Фирма Teletype Corporation была частью AT&T, и производимые ею телетайпы широко использовались для отправки сообщений, а затем и для подключения к компьютерам в системе Белла и в других местах. Все набираемое на телетайпе отправлялось на компьютер, а ответная информация печаталась (в верхнем регистре) на длинном рулоне бумаги. На фото можно заметить верхнюю часть бумажных рулонов.

Возможно, причина коротких имен у многих команд в Unix в том, что набор текста на телетайпе модели 33 требовал значительных физических усилий и происходил очень медленно.

Кто-то даже сконструировал экспериментальную «портативную» модель 33, поместив клавиатуру и принтер в напоминающий чемодан контейнер, который теоретически можно было носить с собой, но недалеко, так как он весил 25 кг. Колесиков у этого «чемодана» не было. Он подключался к удаленному компьютеру через телефонную линию и встроенный акустический соединитель: телефонная трубка вставлялась в пару резиновых гнезд, и компьютер преобразовывал данные в звук и обратно, что больше напоминало факс. Пару раз я таскал один из этих терминалов домой, и должен сказать, что называть его портативным — явное преувеличение.

Ситуация заметно улучшилась после появления телетайпа модели 37. Он позволял набирать как строчные, так и прописные буквы и работал несколько быстрее (15 символов в секунду, а не 10), хотя печатать на нем все еще было трудно. Расширенная клавиатура позволяла набирать математические символы, что было полезно как для патентных заявок, так и для наших собственных технических документов. Кроме того, бумагу можно было перемещать вперед и назад с шагом в половину строки, что давало возможность добавлять надстрочные и подстрочные индексы.

Подача бумаги тоже была нелегким делом; требовались нешуточные усилия, чтобы загрузить новую партию фальцованной бумаги. Боб Моррис однажды отправил Джо Оссанне по электронной почте сообщение, состоящее из 100 символов обратного перевода строки. При попытке прочитать это сообщение модель 37 начала выплевывать бумагу на пол.

Первые несколько лет Боб занимал кабинет напротив моего. В Bell Labs «Robert Morris» — это достаточно распространенное сочетание. Более того, к Бобу как-то даже приходил посетитель с таким именем. Поэтому Боб часто получал чужую почту, которую отправлял обратно, объясняя, что он не тот Моррис. Однажды ему начали носить тщательно проработанный проект из какой-то другого отдела компании с просьбой: «Пожалуйста, подпишите и верните». Все попытки отправить послание назад терпели неудачу, поэтому Боб подписал его и после этого уже больше никогда не видел.

3.2. КОМНАТА UNIX

Несмотря на то что у каждого сотрудника отдела исследований был личный кабинет, большая часть разработок происходила в помещении, которое называлось «комната Unix». Его местоположение несколько раз менялось, но это всегда было место, где можно было отдохнуть, узнать новости, поделиться идеями или просто пообщаться.

Самая первая комната Unix находилась на четвертом этаже здания 2, где установили PDP-7. Но это было ненадолго, потом много лет мы собирались на шестом этаже здания 2 в комнате 2C-644. Офисных этажей в этом здании было всего пять, а шестой этаж представлял собой технический коридор: грязный, плохо освещенный и полный складских помещений с пыльным заброшенным оборудованием.

На одном конце открытая площадка с автоматами, в которых продавались ужасный кофе и почти несъедобное печенье, служившие топливом для ночного программирования. Еще было несколько замкнутых пространств, одно из которых, по крайней мере в течение десяти лет, служило комнатой Unix. Именно там стоял компьютер PDP-11; там сделана фотография Кена и Денниса (рис. 3.1). Несколько столов, стульев и терминалов превратили это место в прекрасную общую рабочую зону.

Из людей, не работавших в подразделении 1127, одним из первых поклонников Unix стал выдающийся физик-теоретик, ныне покойный, которого я буду называть М. Л. Он горел желанием работать в Unix, видел перспективы применения компьютеров в физике, кроме того, был добрым и щедрым человеком. Но при этом говорлив был необычайно. Когда он открывал рот, его было невозможно остановить. Он мог часами вещать в режиме монолога. В итоге в матовом покрытии стекла двери в комнату Unix кто-то процарапал маленькую дырочку, чтобы, прежде чем войти, можно было заглянуть внутрь и узнать, есть ли там М. Л. Мы называли это Л-отверстием.

Потом комната Unix переместилась в комнату 2C-501, которая находилась на пятом этаже у лестницы 9, за углом от моего кабинета. Туда начали покупать различные кофеварки. Сначала обычные кувшины для кофе с нагревателем, которые поддерживали тепло, пока кофе, а то и сам кувшин не сгорал (что происходило достаточно регулярно). Затем стали закупать более дорогие кофемолки и кофе-машины (рис. 3.2). Последняя кофе-машина стоила около трех тысяч долларов. Если мои источники не обманывают, обитатели комнаты Unix собирали на нее деньги вскладчину, а руководство платило за кофе.

В этой комнате было весело, там всегда что-то происходило. Некоторые предпочитали работать только там, забывая про свои кабинеты. Другие заходили несколько раз в день, чтобы выпить кофе и поболтать. Комната Unix, как ничто другое, помогала следить за тем, что делают коллеги, и поддерживала чувство общности.

Я считаю, что в Bell Labs хорошо умели работать с пространством. Отдельные кабинеты, хотя и стоят дороже опенспейсов, предоставляют людям место, где можно хранить книги и бумаги и сосредоточиться без вечного шума на заднем плане. Если требуется тщательно обдумать что-нибудь или побеседовать с глазу на глаз, достаточно закрыть дверь. Я много работал в опенспейсах и могу с уверенностью сказать, что они гибельно действуют на способность к сосредоточению. В Bell Labs у сотрудников были как личные кабинеты, так и общее рабочее пространство.

По вечерам сотрудники Bell Labs могли работать из дома. Много лет у меня была выделенная телефонная линия (в конце концов, именно этим занималась компания AT&T), позволяющая подключаться к Unix-машинам в Мюррей-Хилл, что позволяло работать по вечерам и выходным. Неожиданным бонусом оказался специальный код доступа для совершения

Рис. 3.2. Кофе-машина и кофемолки в комнате Unix

нетарифицируемых междугородных телефонных звонков в любую точку страны. В то время это было огромной привилегией, ведь междугородные звонки стоили довольно дорого. О том, как мы ее получили, рассказал Кен Томпсон:

> Джо Оссанна решил, что мы заслужили домашние телефонные линии и телетайпы. Он придумал форму для их заказа и копии этой формы положил на склад канцелярских товаров. Себя Джо назначил ответственным за согласование, а затем написал несколько заявок от ключевых фигур в группе Unix. Дальше достаточно было нескольких телефонных звонков, и к Джо начали поступать заполненные формы, которые он тут же одобрял. Фактически он предоставил нам этот бонус, просто изобретя форму для его заказа.

В 1985 году на должность начальника подразделения 1127 назначили Питера Вайнбергера. По этому поводу профессиональный фотограф снял его для газеты Bell Labs *Bell Labs News* (которую все называли *Bell Labs Good*

News, так как там публиковались только хорошие истории). Оставив отпечаток этого снимка (рис. 3.3) в комнате Unix, Питер совершил серьезную тактическую ошибку.

Рис. 3.3. Фотография Питера (из архива Джерарда Хольцмана)

Вскоре его изображение было повсюду, иногда пропущенное через недавно разработанный логотип AT&T (рис. 3.4). Как рассказывал Джерард Хольцман:

> Через несколько недель после того, как AT&T представила новый корпоративный логотип, Том Дафф сделал логотип Peter (рис. 3.5), который стал символом нашего центра. Роб Пайк заказал футболки с этим символом, а Кен Томпсон — кофейные кружки.

Рис. 3.4. Логотип AT&T, который сотрудники иногда называли «Звезда смерти»

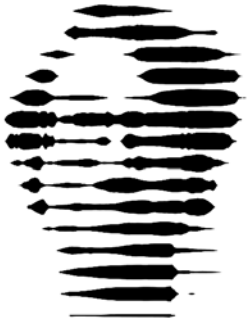


Рис. 3.5. Питер, пропущенный через логотип AT&T
(из архива Джерарда Хольцманна)

Много лет подряд лицо Питера появлялось в самых неожиданных местах. В каждой ячейке организационной структуры Bell Labs, сформированное из маленьких круглых магнитов на металлической стене рядом с лестницей, в виде оттисков на новом бетонном полу и на микропроцессорных кристаллах. И что самое примечательное, ночью 16 сентября 1985 года кто-то нанес логотип с его лицом на одну из водонапорных башен Bell Labs (рис. 3.6).

Ходили разные слухи о том, кто это сделал, но даже сейчас, более трех десятилетий спустя, точной информации нет. Был подан запрос на возмещение стоимости краски, который отклонили. А через несколько дней логотип на башне закрасили по приказу администрации, которая явно не разделяла нашего чувства юмора.

Целиком историю логотипов с лицом Питера можно найти на сайте spinroot.com/pjw. Сайт этот создал и поддерживает Джерард, который вместе с Робом Пайком вносил многочисленные доработки в исходное фото.

Bell Labs славились своей неформальной атмосферой, но ближе к середине 1980-х там ввели новое правило: обязательное ношение бейджей. Несомненно, это была разумная мера предосторожности, позволяющая отсекать посторонних, но популярности она не снискала. В знак протеста один коллега приклеил бейдж ко лбу суперклеем, а другой начал цеплять его к волосам на груди, показывая только по требованию.

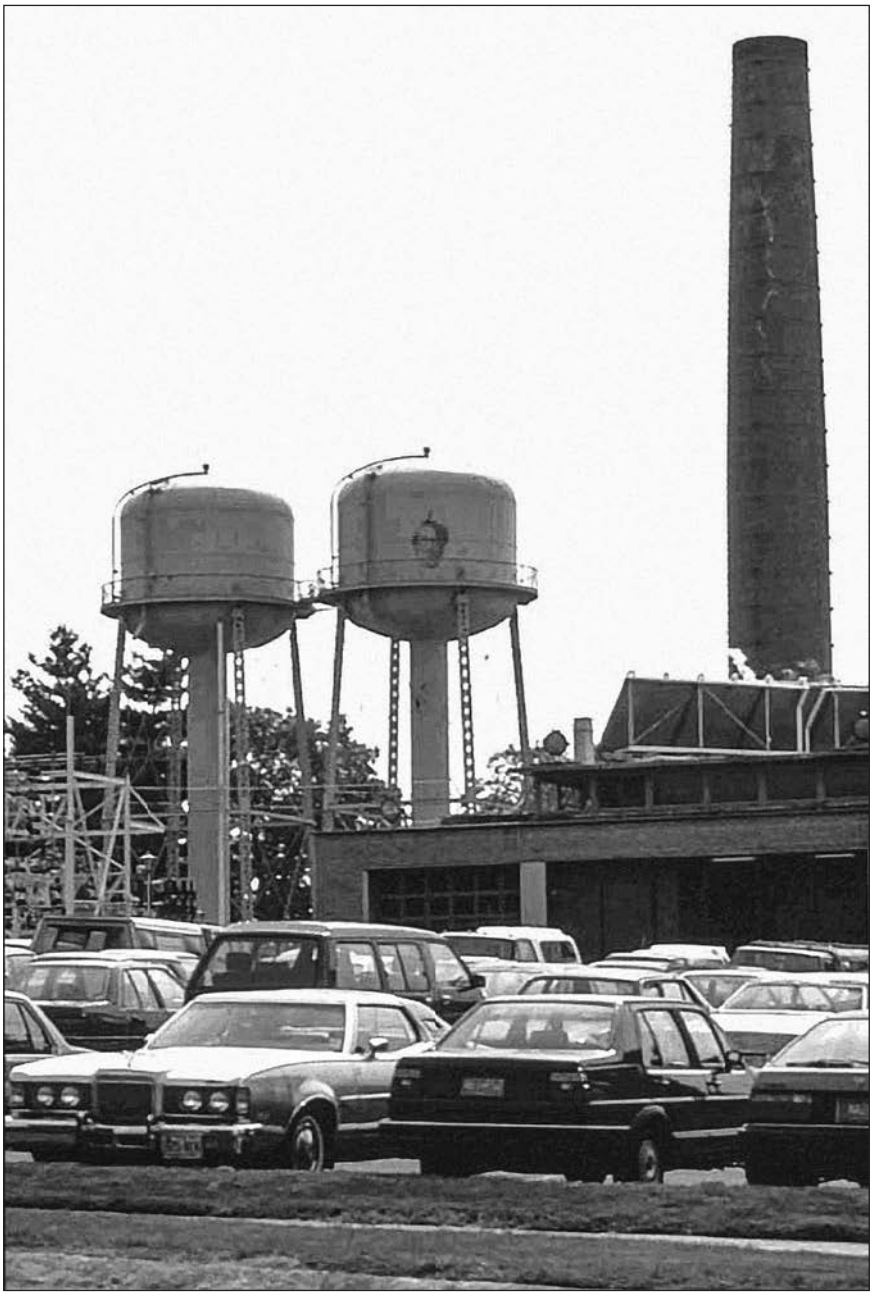


Рис. 3.6. Питер на водонапорной башне, 1985 год (из архива Джерарда Хольцманна)

Защитных элементов бейджи не имели. Они представляли собой всего лишь шаблонную картинку. Поэтому мы сфабриковали персонажа Грейса Р. Эмлина, у которого была своя учетная запись `gre` и собственный бейдж (рис. 3.7), а время от времени он даже появлялся в официальных списках и публикациях.

Рис. 3.7. Сотрудник подразделения Грейс Эмлин

Себе я сделал бейдж с изображением Микки Мауса (рис. 3.8) и регулярно носил его. Даже в тот день в Холмделе, Нью-Джерси, когда в Bell Labs с целью продвижения Windows 3.0 приезжал Билл Гейтс. Никто не обратил на это внимания.

На фотографиях 3.9 и 3.10 показана комната Unix в 2005 году.

Рис. 3.8. Мой бейдж с высоким уровнем защиты информации

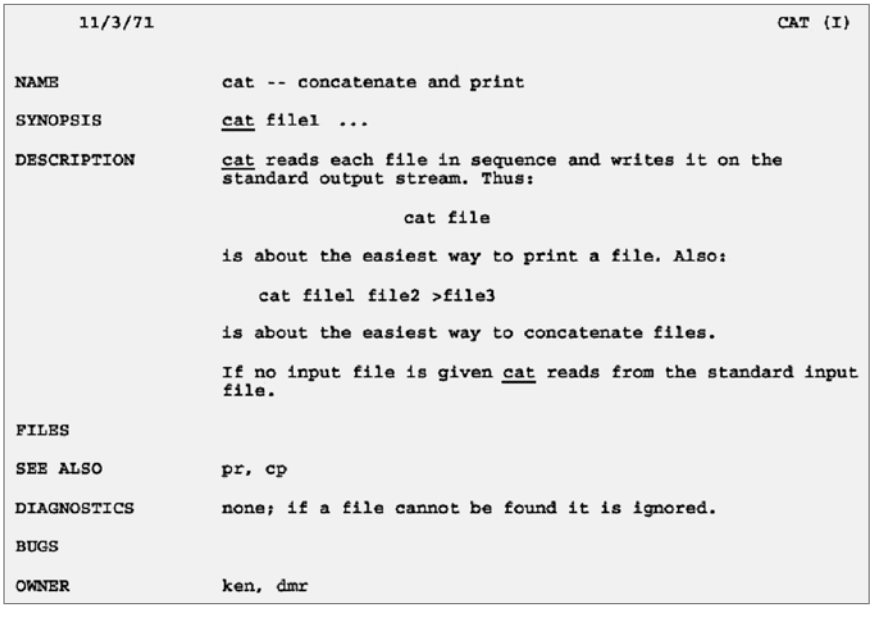
Рис. 3.9. Комната Unix, октябрь 2005 года

Рис. 3.10. Комната Unix, октябрь 2005 года

3.3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ UNIX

Одним из ранних достижений Unix стало электронное руководство во всем известном ныне формате и лаконичном стиле. У каждой команды, библиотечной функции, формата файла и прочего была своя страница с описанием того, что это такое и как этим пользоваться. Например, на рис. 3.11 приведена страница из первого издания для команды `cat`, объединяющей указанные пользователем файлы в стандартный поток, который по умолчанию выводится в пользовательский терминал.

В первых версиях руководства информацию старались умещать в буквальном смысле на одной странице, что сейчас редкость. При этом появился такой новый для того времени раздел, как ОШИБКИ (BUGS). Как своего рода признание того факта, что в программах могут встречаться ошибки или какие-то особенности и недостатки, которые, даже если их сразу не исправляют, должны быть документированы.



```
      11/3/71                                                    CAT (I)


NAME                cat -- concatenate and print

SYNOPSIS            cat file1 ...

DESCRIPTION         cat reads each file in sequence and writes it on the
                    standard output stream. Thus:

                                    cat file

                    is about the easiest way to print a file. Also:

                        cat file1 file2 >file3

                    is about the easiest way to concatenate files.

                    If no input file is given cat reads from the standard input
                    file.

FILES

SEE ALSO            pr, cp

DIAGNOSTICS         none; if a file cannot be found it is ignored.

BUGS

OWNER               ken, dmr
```

Рис. 3.11. Страница руководства, посвященная команде cat(I) из первой редакции Unix

За прошедшие полвека команда `cat` практически не изменилась. Разве что появилось несколько необязательных и, вероятно, ненужных аргументов, меняющих ее поведение. Но она все еще входит в основной набор команд Unix. Современную справочную информацию по ней можно посмотреть, набрав в терминале операционных систем семейства Linux и macOS или в подсистеме Windows для Linux (WSL) команду

```
$ man cat
```

Разумеется, таким же способом можно вывести страницу руководства для самой команды `man`:

```
$ man man
```

3.4. НЕМНОГО О ПАМЯТИ

Читатели помладше могут задаваться вопросом, правильно ли я в своем рассказе указывал размер памяти. Например, компьютеры IBM 7090

и 7094 имели память размером в 32 Кбайт (32 768) 36-битных слов. Оригинальный PDP-7, на котором работал Кен, имел 8 Кбайт (8192) 18-битных слов, то есть примерно одну восьмую памяти 7090. У первой версии PDP-11 было 24 Кбайт оперативной памяти и диск размером полмегабайта. Для сравнения, мой ноутбук Macbook Air 2015 года выпуска имеет 8 Гбайт оперативной памяти (более чем в 330 000 раз больше) и диск на 500 Гбайт (в полмиллиона раз больше) и при этом стоит всего тысячу долларов.

Сегодня гигабайты оперативной памяти и терабайты дискового пространства стоят дешево, имеют компактные размеры и широко распространены. По сравнению с современными реалиями раньше объем памяти был крошечным. Кроме того, в 1960-х и начале 1970-х память производилась по другой технологии.

Для нее использовались массивы крошечных ферритовых сердечников в форме кольца, которые определенным способом соединялись набором проводов. Каждый сердечник можно намагнитить в двух направлениях (по часовой стрелке или против часовой стрелки), что позволяло представить бит информации. Восемь сердечников составляли байт.

Память на магнитных сердечниках стоила очень дорого, так как ее изготовление требовало высококвалифицированного ручного труда. Кроме того, она была громоздкой и тяжелой. На рис. 3.12 показано 16 Кбит (2 Кбайт) памяти, которая в 1971 году стоила примерно 16 000 долларов, то есть примерно по одному доллару за бит.

Часто память оказывалась самым дорогим из комплектующих. Огромная ценность каждого байта дисциплинировала программистов. Все время приходилось думать о количестве используемой памяти. Иногда для размещения программы прибегали к хитрым и рискованным методам программирования.

Операционная система Unix эффективно использовала небольшую доступную в то время оперативную память. Отчасти это заслуга исключительно талантливых программистов, таких как Кен и Деннис, которые знали, как ее сэкономить. Придумывали гениальные способы обеспечения универсальности и единообразия, позволяющие получить как можно больше из как можно меньшего количества кода. Иногда это достигалось искусным программированием, иногда путем совершенствования алгоритмов.

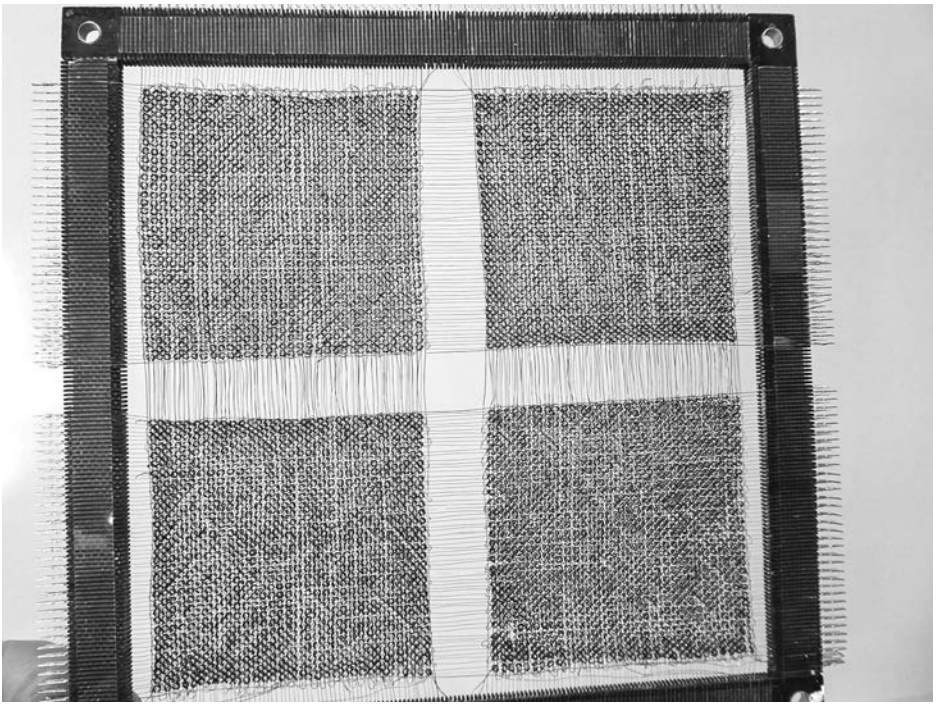


Рис. 3.12. Память на магнитных сердечниках; 16 Кбит, 2 Кбайт (размер примерно 13 см)

Помогало и программирование на языке ассемблера, который в то время обеспечивал более быстрое выполнение инструкций и требовал меньше памяти, чем компиляторы языков высокого уровня. Только к середине 1970-х годов стала доступной по цене новая технология изготовления памяти на полупроводниковых микросхемах и появилась возможность допустить пусть небольшие, но заметные затраты вычислительных ресурсов на языки высокого уровня, такие как Си.

Функции управления распределением памяти, такие как `alloc` и написанная позднее Дугом Макилроем библиотека `malloc`, давали еще один способ максимально задействовать ограниченный ресурс. Естественно, требовалась крайняя осторожность, потому что малейшая ошибка могла вызвать сбои в работе программ (я бы добавил, что это актуально до сих пор). Нерациональное распределение памяти остается одной из основных причин ошибок в программах на языке Си.

Критичные сбои программы операционная система фиксировала и создавала для помощи программисту файл с содержимым основной памяти, той, что находилась на магнитных сердечниках (magnetic cores). Именно отсюда возникло словосочетание *дамп памяти* (core dump), которое применяется до сих пор, хотя памятью на магнитных сердечниках давно уже никто не пользуется. Файл с этим дампом так и называется — `core`.

3.5. БИОГРАФИЯ: ДЕННИС РИТЧИ

Это жизнеописание Денниса Ритчи взято из воспоминаний, написанных мной в 2012 году для Национальной инженерной академии США.

Деннис (рис. 3.13) родился в сентябре 1941 года. Его отец Алистер Ритчи много лет проработал в Bell Labs в Мюррей-Хилл. Деннис учился в Гарварде, где получил степень бакалавра по физике и прикладной математике. В 1968 году он защитил докторскую диссертацию по теме «Подрекурсивные иерархии функций». Неспециалисту непросто понять, что это такое, а я в данном случае неспециалист. На рис. 3.14 показан фрагмент страницы из его диссертации. Сам он про свой карьерный путь рассказывал так:

> Студенческий опыт показал, что я недостаточно умен, чтобы стать физиком, а вот компьютеры — это круто. В процессе обучения в аспирантуре я понял, что недостаточно умен, чтобы стать экспертом в теории алгоритмов, и что процедурные языки мне нравятся больше функциональных.

Как однажды выразился создатель языка C++ Бьёрн Страуструп, «если бы Деннис решил потратить это десятилетие на сложные математические материи, операционная система Unix не имела бы шанса на жизнь».

Деннис несколько раз бывал в Bell Labs на летних стажировках и окончательно стал членом технического персонала в Исследовательском центре компьютерных наук в 1967 году. Первые несколько лет он работал над операционной системой Multics. Как я уже отмечал, этот проект оказался чересчур масштабным, и в 1969 году, как только стала очевидной недостижимость поставленных целей, Bell Labs его закрыли. А у Кена, Денниса и их коллег остался опыт работы с инновационным проектированием операционной системы, понимание того, как реализованы языки программи-

Рис. 3.13. Деннис Ритчи, примерно 1981 год (из архива Джерарда Хольцманна)

Now if $\beta = \omega^m b_m + \cdots + \omega^{n+1} b_{n+1} + \omega^n b_n + \cdots + \omega^0 b_0$, let $\beta' = \omega^m b_m + \cdots + \omega^{n+1} b_{n+1}$. Then $\beta + \omega^n(q + b + 5\underline{m} + 1) = \beta' + \omega^n(q + b + b_n + 5\underline{m} + 1)$ and furthermore β' is the least ordinal with this property. Thus by (3.2),

$$
\begin{aligned}
T_{\underset{\sim}{P}}(\bar{x}_k) &\le f_{\beta'+\omega^{n+1}}(q + b + b_n + 5\underline{m} + 1) \\
&= f_{\beta+\omega^{n+1}}(q + b + b_n + 5\underline{m} + 1) && \text{by definition of } \beta' \\
&\le f^{(q+b+b_n+1)}_{\beta+\omega^{n+1}}(5\underline{m}) && \text{by (3.4.v)} \\
&\le f^{(q+b+b_n+1)}_{\beta+\omega^{n+1}} f_1^{(3)}(\underline{m}) && \text{by (3.4.ii)} \\
&\le f^{(q+b+b_n+4)}_{\beta+\omega^{n+1}}(\underline{m}) && \text{by (3.4.vii)}
\end{aligned}
$$

But even if $\underline{m} = 0$, $T_{\underset{\sim}{P}}(\bar{x}_k) = 2 \le f^{(q+b+b_n+4)}_{\beta+\omega^{n+1}}(\underline{m})$ by (3.4.v). Since $t_{n+1}(\beta) = t_n(\beta) + b_n = b + b_n$, the lemma is proved, concluding Case 4.

Рис. 3.14. Выдержка из диссертации Денниса Ритчи (фото предоставлено Музеем компьютерной истории)

рования высокого уровня и возможность начать с нуля, поставив гораздо более скромную цель. В результате всего этого появились Unix и язык программирования Си.

Появление языка Си датируется началом 1970-х. Основой для него послужил опыт Денниса в высокоуровневых языках, которые использовались для написания Multics. Но пришлось все очень сильно урезать из-за ограниченной мощности большинства компьютеров того времени. Для поддержки компилятора сложного языка элементарно не хватало памяти и вычислительных ресурсов. Этот вынужденный минимализм не мешал работе Кена и Денниса, которые предпочитали простые унифицированные механизмы. Хорошо подходил для существующего компьютерного оборудования и язык Си. Было понятно, как написать на нем хороший, эффективно работающий код.

В результате всю ОС удалось создать на языке высокого уровня. К 1973 году произошло преобразование исходной версии Unix, написанной на ассемблере, в версию на Си, что сильно облегчило обслуживание системы и внесение в нее изменений. Кроме того, появилась возможность еще одного гигантского шага: перенос операционной системы с исходного компьютера PDP-11 на компьютеры с другой архитектурой. Поскольку большая часть системного кода была написана на Си, портирование системы сводилось к переносу компилятора этого языка.

Деннис превосходно разрабатывал техническую документацию. У него был собственный элегантный стиль, он умел прекрасно выражать мысли и обладал неподражаемым остроумием. Мы вместе написали книгу «Язык программирования Си», увидевшую свет в 1978 году. В 1988 году вышло второе издание. А за прошедшие годы книга была переведена на более чем два десятка языков. Основную часть книги составил оригинальный справочник по языку Си, написанный Деннисом, который сформировал основу стандартов ANSI и ISO для этого языка. Без сомнения, успех языка Си и операционной системы Unix можно частично объяснить писательским мастерством Денниса.

За работу над языком Си и Unix Деннис вместе с Кеном Томпсоном получил множество наград и премий. В том числе премию Тьюринга (1983), Национальную медаль США в области технологий (1999), Премию Японии за достижения в области информации и средств связи (2011). В 2019 году уже посмертно они были введены в Национальный зал славы изобретателей.

В течение многих лет успешно избегавший любой руководящей роли, Деннис в конце концов уступил и стал руководителем отдела программных систем, где отвечал за группу, занимавшуюся разработкой операционной системы Plan 9. Отошел от управления и официально вышел на пенсию он в 2007 году, но продолжал приходить в Bell Labs почти каждый день до своей смерти в октябре 2011 года.

Деннис был скромным и щедрым и всегда отдавал должное другим, преуменьшая свой вклад. Типичный пример можно увидеть в разделе «Благодарности» в написанной им в 1996 году ретроспективе эволюции Unix:

> По большому счету читатель не ошибется, если будет воспринимать любое местоимение «мы» как «Томпсон с некоторой моей помощью».

Деннис умер в октябре 2011 года. Слова его сестры и братьев до сих пор хранятся на его домашней странице на сайте Bell Labs www.bell-labs.com/usr/dmr/www.

> Как родственники Денниса мы, Линн, Джон и Билл Ритчи, от имени всей нашей семьи хотели бы рассказать, как глубоко мы тронуты, удивлены и благодарны за все добрые слова о Деннисе, которые мы продолжаем читать. Мы можем подтвердить то, что слышим снова и снова.
>
> Денис был неизменно добрым, милым, скромным и щедрым братом и, конечно, фанатом своего дела. У него было тонкое чувство юмора и резкая оценка вещей, выходящих за пределы здравого смысла, хотя его взгляд на мир был полностью лишен цинизма или злобы.
>
> Нам ужасно грустно, что мы потеряли его, но мы невыразимо растроганы тем, какой след он оставил в этом мире, и насколько хорошо смогли понять не только его достижения, но и его великодушную личность.
>
> Линн, Джон и Билл Ритчи

ГЛАВА 4

ШЕСТАЯ РЕДАКЦИЯ (1975)

Количество установок Unix выросло до 10, и ожидается дальнейший рост.

Вторая редакция руководства для программистов Unix, июнь 1971 года

Количество установок Unix превысило 50, и ожидается дальнейший рост.

Пятая редакция руководства для программистов Unix, июнь 1974 года

Если ориентироваться по датам, указанным в руководстве, первая редакция Unix была запущена к концу 1971 года. Следующие несколько лет примерно каждые полгода выходило новое издание руководства, каждый раз с описанием нового функционала, новых инструментов и новых языков. Шестая редакция операционной системы, руководство по которой датируется маем 1975 года, стала первой, вышедшей за пределы Bell Labs и сильно повлиявшей на окружающий мир.

Описание Unix впервые было дано в статье Денниса Ритчи и Кена Томпсона «Система с разделением времени Unix», написанной для организованного Ассоциацией вычислительной техники в октябре 1973 года Четвертого

симпозиума по принципам операционных систем. В июле 1974-го с небольшими изменениями эту статью переиздали в журнале *Communications of the Association for Computing Machinery*. В кратком предисловии было перечислено удивительное количество хороших идей:

Unix — универсальная многопользовательская интерактивная операционная система для компьютеров фирмы Digital Equipment Corporation PDP-11/40 и 11/45. Она предлагает ряд функциональных возможностей, редко встречающихся даже в более крупных операционных системах. В том числе:

1) иерархическую файловую систему, включающую отсоединяемые тома;
2) совместимые файлы, устройства и межпроцессный обмен данными;
3) возможность запускать асинхронные процессы;
4) системный командный язык, выбираемый пользователем;
5) более ста подсистем, включающих десяток языков.

Что же представляли собой эти функциональные возможности, «редко встречающиеся даже в более крупных операционных системах», и каково было их значение? Сейчас мы поговорим об этом более подробно. Те, кого не очень интересуют технические детали, могут бегло пролистать эту главу. Самую важную информацию я даю в начале каждого раздела, а идущий ниже подробный разбор можно пропустить.

4.1. ФАЙЛОВАЯ СИСТЕМА

Файловой системой (file system) называется набор правил, определяющий способ организации, хранения и именования данных на носителях информации, например дисках, которые долгое время представляли собой сложные механические устройства, основанные на принципе магнитной записи. Сегодня все чаще встречаются твердотельные диски и флеш-накопители USB на основе микросхем памяти, не имеющие движущихся частей.

Абстрактное представление файловой системы можно увидеть в Проводнике в Windows и Finder в macOS. В их основе значительное количество программного обеспечения, которое осуществляет управление данными на физических носителях, следит за тем, где что находится, отвечает за доступ на чтение и запись и поддерживает все в согласованном состоянии.

До появления Multics в большинстве операционных систем в лучшем случае имелись запутанные и нестандартные файловые системы. Файловая система Multics стала более универсальной, упорядоченной и мощной, но и более сложной. Разработанная Кеном файловая система Unix, сохранив достижения Multics, получилась намного проще. Ее понятная и элегантная архитектура постепенно стала широко использоваться и эмулироваться.

Файл в Unix представляет собой просто последовательность байтов. Структура файла и порядок его содержимого определяются только программами для его обработки. Файловая система никак не касается того, что внутри. Это означает, что любая программа может читать любой файл и делать в него запись. Сейчас такой подход кажется само собой разумеющимся, но его не признавали в более ранних системах, где на формат данных в файлах и способы их обработки программами иногда накладывались различные ограничения. Вот пример от Дуга Макилроя:

> Обычно исходный код принадлежит к собственному типу, отличному от данных. Компиляторы читают исходный код, скомпилированные программы умеют считывать и записывать «данные». Соответственно, программы на Фортране создавались и проверялись совсем не так, как все прочие файлы. Для них существовали свои способы редактирования и вывода. Это исключало возможность использования какой-либо программы для генерации (или даже простого копирования) кода на Фортране.

В Unix подобных различий не существует: любая программа может обрабатывать любой файл. Разумеется, бывают ситуации, когда применять программу к файлу бессмысленно. Например, невозможно скомпилировать файл, содержащий код на Фортране, компилятором для Си. Но это никак не связано с операционной системой.

Файлы Unix распределены по каталогам. (В других операционных системах их часто называют папками.) Каталог Unix — это также файл в файловой системе, просто его содержимое обрабатывается самой системой, а не пользовательскими программами. Каталог содержит информацию о файлах, которые, в свою очередь, могут быть каталогами.

Запись в каталоге Unix включает имя файла, права доступа, размер файла, дату и время его создания и изменения, а также информацию о том, где находится его содержимое. В каждом каталоге есть две специальные записи:

«.» (сам каталог) и «..», родительский каталог. Вершина иерархии — корневой каталог /. Для доступа к любому файлу нужно спуститься вниз от корневого каталога, а в корневой каталог можно подняться из любого места, последовательно выходя в «..» родительские каталоги. Например, текст этой книги располагается по адресу /usr/bwk/book/book.txt. Система поддерживает понятие текущего каталога, что позволяет указывать имена файлов относительно текущего местоположения в файловой системе вместо абсолютного пути от корня.

Поскольку любой каталог может содержать подкаталоги, формально файловая система допускает произвольную глубину. Структура из вложенных каталогов и файлов называется иерархической файловой системой. Опять же, несмотря на их очевидные преимущества, иерархические файловые системы не имели широкого распространения до операционной системы Multics. В частности, накладывались ограничения на глубину вложенности; например, в CTSS допускалось всего два уровня.

4.2. СИСТЕМНЫЕ ВЫЗОВЫ

Операционная система предоставляет набор служб для работающих в ней программ. Это запуск и остановка программ, чтение или запись информации в файлы, доступ к устройствам и сетевые подключения, сообщение даты и времени и многое другое. Эти реализованные в операционной системе службы доступны из запущенных программ через механизм, называемый *системными вызовами* (system calls).

По сути, системные вызовы и есть операционная система, так как именно они определяют, какие службы она предоставляет. Возможны разные независимые реализации наборов системных вызовов, например, в случае с версиями Unix и Unix-подобных систем. Другая ОС, скажем Windows, может предоставлять программное обеспечение для преобразования системных вызовов Unix в собственные системные вызовы. При этом всегда будут существовать системные вызовы, уникальные для конкретной ОС, даже если она похожа на Unix.

В первой редакции Unix было чуть больше 30 системных вызовов, половина которых была связана с файловой системой. Поскольку файлы содержали только неинтерпретируемые байты, базовый интерфейс файловой систе-

мы был очень простым. Существовало всего пять системных вызовов: для открытия или создания файла, для чтения или записи в него и для его закрытия. Доступ к ним осуществлялся из программы на Си, а функции вызывались с помощью таких операторов, как:

```
fd = creat(filename, perms)
fd = open(filename, mode)
nread = read(fd, buf, n)
nwrite = write(fd, buf, n)
status = close(fd)
```

Системный вызов `creat` создает новый файл и устанавливает права на доступ к нему. Обычно эти права включают или исключают возможность чтения, записи и выполнения для пользователя, для группы пользователей и для всех остальных. Этот сравнительно небольшой механизм из девяти битов позволяет хорошо контролировать права доступа. Системный вызов `open` открывает существующий файл; `mode` указывает, открывается ли он на чтение или на запись, а атрибут `filename` задает путь к файлу в иерархической файловой системе.

Значение `fd`, появляющееся как результат вызовов `open` и `creat`, называется *дескриптором файла* (file descriptor). Это небольшое неотрицательное целое число, которое в дальнейшем используется при операциях чтения из файла и записи в него. Системные вызовы `read` и `write` пытаются передать `n` байтов либо из файла, либо в файл. Функции `nread` и `nwrite` при этом возвращают количество фактически переданных байтов. Возврат отрицательного значения (обычно –1) для всех системных вызовов указывает на какую-то ошибку.

Между прочим, системный вызов `creat` действительно пишется именно так: никаких причин для этого нет, просто так захотел Кен Томпсон. Роб Пайк однажды спросил Кена, что бы он поменял, если бы ему пришлось снова создавать Unix. Каким был его ответ? «Я бы написал `creat` как полагается — с `e` на конце».

Еще одним нововведением Unix стало добавление в файловую систему в виде файлов периферийных устройств: дисков, терминалов и прочего. Диски — это те самые отсоединяемые тома, которые упоминались при перечислении функциональных особенностей. Для доступа к устройствам используются те же системные вызовы, что и для доступа к файлам, соответственно, один код может управлять файлами и устройствами. Разумеется,

все не настолько просто, ведь реальные устройства обладают различными дополнительными свойствами, которые необходимо обрабатывать. Поэтому существуют дополнительные системные вызовы, учитывающие специфические характеристики, особенно в случае терминалов. Эта часть системы уже не такая красивая.

Также существуют системные вызовы для задания положения внутри файла, определения его статуса и прочего. За прошедшие полвека их то и дело пытались украшать, а иногда и улучшать, но проста и удобна в использовании именно базовая модель.

Современному читателю сложно представить, насколько все было сильно упрощено. В ранних операционных системах пользователи видели все компоненты устройств. Требовалось знать все об именах дисков, их физической структуре, например о количестве дорожек и цилиндров, а также о том, как на них организованы данные. Стив Джонсон напомнил мне, какой неудобной была в то время подсистема разделения времени на главном компьютере Honeywell:

> Чтобы создать файл на диске компьютера Honeywell, нужно было войти в подсистему, а затем ответить примерно на восемь вопросов. Про начальный размер файла, про его максимально допустимый размер, про его имя, про устройство, про то, кто имел право его читать и кто имел право на запись в него, и т. п. Отвечать требовалось в интерактивном режиме. Если после ответов на все вопросы операционная система обнаруживала, что вы что-то некорректно ввели, создавать файл она отказывалась. Это означало, что нужно снова войти в подсистему и снова отвечать на вопросы. Неудивительно, что когда файл наконец-то создавался, система писала УСПЕШНО (SUCCESSFUL!).

По примеру Multics в Unix начали скрывать всю эту несущественную информацию. Файлы превратились просто в наборы байтов. Пользователь указывал, что это за байты, а операционная система следила за их хранением и извлечением, не показывая пользователям все свойства устройства.

4.3. КОМАНДНАЯ ОБОЛОЧКА

Оболочка — это программа, позволяющая запускать другие программы. Это основной интерфейс взаимодействия пользователей и операционной

системы. При входе в систему Unix клавиатура подключается к работающему экземпляру оболочки. Теперь можно вводить команды, обычно по одной. Выполнив одну команду, оболочка готова к вводу следующей. Вот пример сеанса, где знак доллара $ — приглашение на ввод. Введенные пользователем команды выделены вот таким шрифтом.

```
$ date          выводятся дата и время
Fri Oct 18 13:09:00 EDT 2019
$ ls            список содержимого каталога
book.pdf
book.txt
$ wc book.txt подсчет строк, слов и символов в файле book.txt
    9918 59395 362773 book.txt
$ cp book.txt backup.txt копирование book.txt в резервный файл
```

Важно отметить, что оболочка (ее идея взята из Multics) — это обычная пользовательская программа, а не неотъемлемая часть операционной системы. (В списке функциональных возможностей это «системный командный язык, выбираемый пользователем».) Пользовательскую программу легко заменить другой, поэтому в Unix существует так много разновидностей командных интерпретаторов. Если не нравится, как работает одна оболочка, всегда можно выбрать другую или даже написать собственный вариант.

При этом все оболочки Unix предоставляют одинаковую базовую функциональность, как правило, с одинаковым синтаксисом. Наиболее важная функциональность — запуск программ. Во всех оболочках при указании имен файлов можно применять подстановочные символы, такие как *. Эти символы расширяют список имен файлов, соответствующих шаблону. Например, чтобы запустить утилиту wc (word count — подсчет слов) для подсчета строк, слов и символов во всех файлах текущего каталога, имена которых начинаются с book, достаточно написать

```
$ wc book*
```

Оболочка примеряет шаблон book* к именам всех файлов в текущем каталоге, выбирает из них те, которые начинаются с book, и запускает утилиту wc с этими именами в качестве аргументов. При этом команда wc не знает, что список имен файлов задан с помощью шаблона. Важно, что проверка на соответствие шаблону выполняется оболочкой, а не отдельными программами. Например, в операционной системе MS-DOS от Microsoft в течение многих лет этот подход не работал, поэтому некоторые программы умели

проводить подобную проверку, а некоторые нет; пользователи не могли рассчитывать на согласованное поведение.

Другая важная возможность командной оболочки — перенаправление ввода и вывода. Программу, которая читает со стандартного ввода (по умолчанию это терминал), можно заставить читать из входного файла:

```
$ имя_программы <входной_файл
```

а стандартный вывод программы (по умолчанию это тоже терминал) можно перенаправить в файл с помощью команды

```
$ имя_программы >выходной_файл
```

Если выходного файла не существует, эта команда его создает. При этом сама программа не знает о перенаправлении ввода и вывода.

Этот универсальный механизм применяется оболочкой, а не отдельными программами. Он проще альтернативного подхода, при котором имена входного и выходного файлов нужно задавать в виде отдельных параметров:

```
$ имя_программы in=входной_файл out=выходной_файл
```

Сценарий оболочки представляет собой сохраненную в файле последовательность команд. Запущенный в экземпляре командной оболочки файл сценария по очереди запускает все перечисленные в файле команды, как если бы они набирались вручную:

```
$ sh <файл_сценария
```

Сценарий инкапсулирует последовательность команд. Например, при работе над этой книгой я запускал последовательность простых программ для поиска орфографических и пунктуационных ошибок, неправильных команд форматирования и прочего. Я мог снова и снова набирать команды запуска этих программ. Но вместо этого я поместил всю последовательность в файл сценария `check`, что дало мне возможность выполнять все варианты проверки с помощью всего одной команды. Есть у меня и другие сценарии: для вывода книги на экран и создания резервной копии.

Такие сценарии стали новыми командами Unix. Это обычное применение сценариев оболочки, избавляющее от необходимости раз за разом вводить часто повторяющиеся последовательности команд. Я до сих пор пользуюсь

некоторыми сценариями, которые написал 30, а то и 40 лет назад, и это вполне обычная ситуация для людей, которые давно работают в Unix.

Для превращения программ командной оболочки в эквивалент скомпилированных программ оставалось сделать так, чтобы файл, помеченный как исполняемый, передавался в оболочку для выполнения. Как только это было сделано, сценарии оболочки стало можно запускать вот такими командами:

```
$ check book.txt
```

Заменой скомпилированных программ сценарии оболочки не стали, однако это важная часть инструментария программиста, применяемая как для личного пользования, так и для решения более крупных задач. Если вы обнаруживаете, что раз за разом набираете одну и ту же последовательность команд, поместите их в сценарий, и рутинная работа будет автоматизирована. Слишком медленный сценарий оболочки можно переписать на другом языке. Дополнительные возможности, которые дают сценарии оболочки, мы рассмотрим в следующем разделе, посвященном конвейерам.

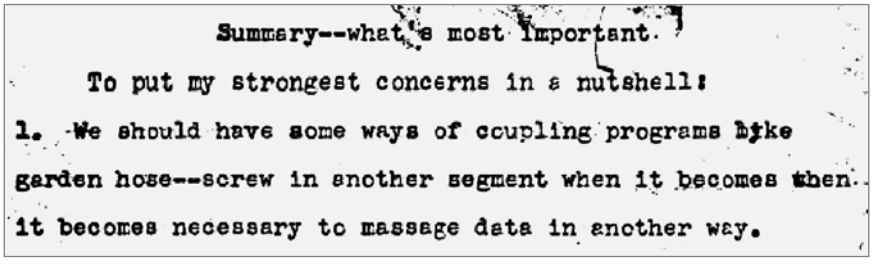
4.4. КОНВЕЙЕРЫ

Конвейеры (pipes) стали, пожалуй, одним из самых ярких изобретений в Unix. Это предоставляемый операционной системой и доступный через оболочку механизм, соединяющий вывод одной программы с вводом другой. Его работу обеспечивает операционная система; для создания конвейера применяются простые и естественные команды оболочки. Конвейеры дали новый подход к проектированию и использованию программ.

Идея соединения программ долгое время витала в воздухе. Одно из самых четких изложений этой идеи в контексте Unix дал в 1964 году Дуг Макилрой в служебном документе, где среди прочего он пропагандировал идею объединения программ «как садовый шланг». На рис. 4.1 приведен фрагмент страницы, которая 30 лет провисела на стене моего кабинета в Bell Labs. Обратите внимание на опечатки и ужасное качество. Вот так зачастую выглядели документы, напечатанные на машинке.

Дуг хотел разрешить произвольные соединения программ в своего рода сетку, но было непонятно, как описать неограниченный граф. Возникали

и семантические проблемы: передаваемые из программы в программу данные должны были корректно ставиться в очередь, а неупорядоченное соединение могло привести к внезапному быстрому росту этой очереди. Кроме того, Кен не мог придумать всему этому реальных вариантов применения.


Summary--what's most important.

To put my strongest concerns in a nutshell:

1. We should have some ways of coupling programs like garden hose--screw in another segment when it becomes then. it becomes necessary to massage data in another way.

Текст гласит:
Самое важное.
В двух словах о том, что меня больше всего беспокоит.
1. У нас должны быть способы соединения программ, как садовый шланг, — возможность подсоединяться к другому сегменту, когда возникает необходимость манипулировать данными другим способом.

Рис. 4.1. Идея конвейера, предложенная Дугом Макилроем (1964)

Но Дуг продолжал его теребить, и в один прекрасный день у Кена появилась идея конвейера практически в том виде, как мы ее знаем сегодня. В течение часа он добавил в операционную систему системный вызов `pipe`. По его словам, это была тривиальная задача, так как механизмы перенаправления ввода и вывода уже были реализованы. Затем Кен добавил механизм конвейера к оболочке, протестировал его и назвал результат сногсшибательным.

Символ конвейера выглядит очень просто. Это вертикальная черта `|`, разделяющая пару команд. Например, для подсчета количества файлов в каталоге нужно направить вывод команды `ls` (по одной строке на файл) на вход команды `wc` (подсчет строк):

```
$ ls | wc
```

Часто программу можно воспринимать как фильтр, считывающий данные, как-то их обрабатывающий и записывающий вывод. Для программ, которые выбирают, видоизменяют или подсчитывают что-то на лету, такая аналогия совершенно естественна, но бывают и другие ситуации. Например, команда

`sort` сначала должна прочитать все входные данные и только потом начать с ними работать. Впрочем, это не имеет значения, ее все равно надо воспринимать как фильтр, который можно встроить в конвейер.

За одну ночь Кен и Деннис обновили все команды в системе. Основным изменением стала возможность чтения данных из стандартного потока ввода даже при отсутствии аргументов с именем входного файла. Еще им пришлось добавить поток стандартного вывода ошибок `stderr`. Это позволило отделять сообщения об ошибках от стандартного вывода, чтобы они не передавались дальше по конвейеру. В целом работа оказалась несложной. В большинстве случаев требовалось добавить только удаление посторонних сообщений, чтобы они не загромождали конвейер, и отправку отчетов об ошибках в `stderr`.

Добавление конвейеров вызвало лавину изобретений, которые я хорошо помню. Не могу сказать точную дату, но это случилось во второй половине 1972 года, поскольку во втором издании руководства (июнь 1972 года) конвейеров еще не было, но они появились в третьем (февраль 1973 года).

У каждого в комнате Unix появились прекрасные идеи по решению различных задач методом объединения программ. Одна из моих идей базировалась на команде `who`, которая выводит список подключенных к системе пользователей. Сейчас, в эпоху персональных компьютеров, она не очень актуальна, но в те времена было полезно знать, кто работает в системе одновременно с тобой. Это не только способствовало чувству общности — если возникала проблема, можно было попросить помощи, даже если дело происходило поздно ночью.

Команда `who` построчно выводит подключенных к системе пользователей, утилита `grep` находит все строки, совпадающие с заданным выражением, а команда `wc` подсчитывает количество строк. Вот примеры конвейеров, позволяющих узнать состояние авторизованных пользователей:

```
who                  # кто авторизован в системе?
who | wc             # сколько авторизованных пользователей?
who | grep joe       # вошел ли в систему joe?
who | grep joe | wc  # сколько раз joe входил в систему?
```

Чтобы понять, насколько большим шагом вперед стали конвейеры, попробуем решить последнюю задачу без них, пользуясь только перенаправлением ввода и вывода в файлы. Это будет выглядеть так:

```
who >temp1
grep joe <temp1 >temp2
wc <temp2
```

После этого еще понадобится удалить временные файлы. Конвейер же сокращает код до одной строки, позволяя обойтись без временных файлов.

У Кена любимым примером применения конвейеров стал говорящий калькулятор, основой для которого служил пакет арифметических вычислений `dc` Боба Морриса. Написанная Кеном программа `number` отображала числа в виде слов («127» превращалось в «сто двадцать семь»), они передавались в программу `speak`, которая на основе полученного ввода синтезировала речь. В интервью в 2019 году Кен рассказывал:

> Вы печатаете в программе `dc` *1 2 +*, конвейер переводит все в программу `number`, а затем в программу `speak`, и вы слышите ответ «четыре».
>
> [смех]
>
> Я никогда не успевал по математике.

Конвейеры стали одним из самых крупных вкладов в Unix, но очевидным это сделалось только много лет спустя. Как сказал Деннис в своей статье *The Evolution of the Unix Time-sharing System* 1984 года:

> Гениальность конвейера Unix состоит в том, что он строится из команд, которыми мы постоянно пользуемся. Но для того чтобы увидеть эту возможность и изобрести для нее нотацию, потребовалось немало сил.

4.5. УТИЛИТА GREP

Основная среда взаимодействия пользователей с Unix — командная строка. Для запуска программ пользователи не щелкают мышью по значкам, как в Windows или macOS, а вводят команды. Интерфейс командной строки не так прост, как управление с помощью мыши, но даже в руках не очень опытного пользователя он оказывается более эффективным, так как позволяет автоматизировать многие вещи, что невозможно в случае графи-

ческого интерфейса. В Unix для применения последовательности команд к большому количеству файлов бывает достаточно одной команды.

В Unix всегда был большой набор инструментов командной строки, то есть программ, выполняющих простые, часто встречающиеся задачи. Полдесятка команд используются для управления файловой системой. Например, `ls` позволяет увидеть список файлов в каталоге, так же как это делают приложение Finder на машинах Mac или Проводник в Windows. Команды `cat` и `cp` применяются для копирования файлов различными способами, команда `mv` — для переименования, а `rm` — для удаления файлов. Есть команды для обработки содержимого файлов; например, `wc` подсчитывает различные вещи, команда `sort` осуществляет сортировку файлов, кроме того, есть несколько команд для сравнения файлов и несколько команд для преобразований, таких как замена регистра. Есть команды для выбора фрагментов файла. (Пользователи Unix сразу вспомнят, что это команды `uniq`, `cmp`, `diff`, `od`, `dd`, `tail`, `tr` и `comm`.) Добавьте еще десяток инструментов, не вписывающихся в вышеперечисленные категории, и набор из 20–30 команд, позволяющий легко выполнять все виды основных задач, готов.

Если сравнивать с естественным языком, то все эти инструменты будут глаголами, а файлы, к которым они применяются, — существительными. Зачастую этот язык нерегулярен, у каждой команды есть необязательные аргументы, меняющие ее поведение. Например, команда `sort` обычно выводит строки в алфавитном порядке, но аргументы позволяют осуществить сортировку в обратном порядке, по значению, по определенным полям и т. д.

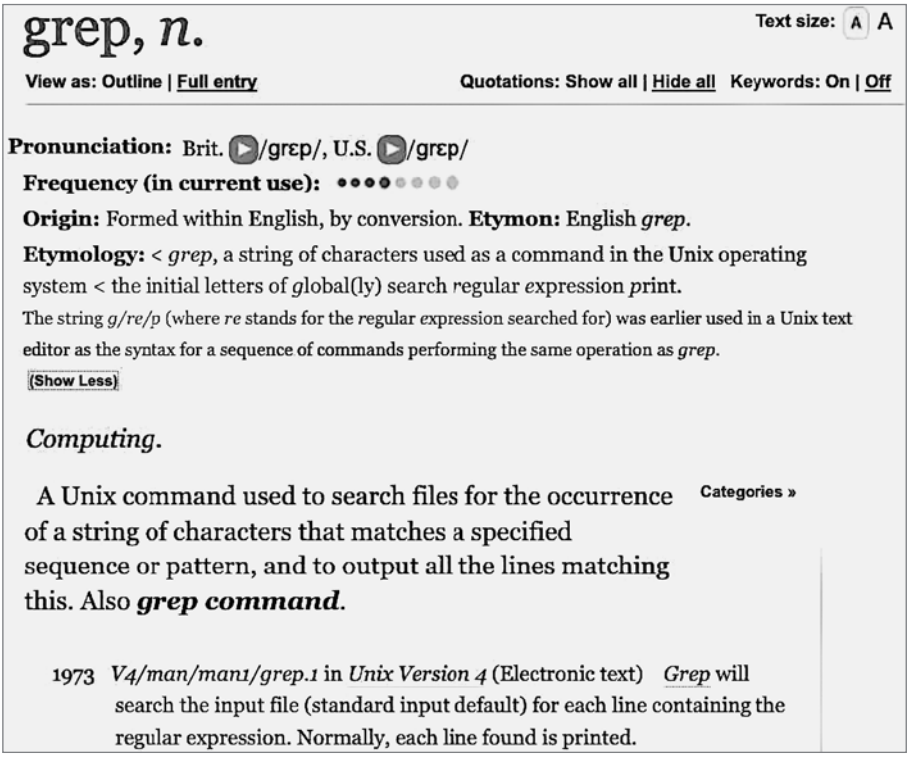
Для уверенной работы в Unix нужно выучить неправильные глаголы, как в обычном языке. Естественно, часто возникают жалобы на исторические отклонения от правил, но попытки логически их обосновать по большей части ни к чему не приводили.

Прототипом инструмента, который заставил воспринимать «инструменты» как нечто большее, чем «программы», стала утилита grep, предназначенная для поиска шаблонов. Автором первой версии был Кен Томпсон. Вот что он сказал о ней в 2019 году:

Я написал эту программу, но в центральный репозиторий не выложил, так как не хотел, чтобы люди думали, будто я им что-то навязываю.

> Когда Дуг Макилрой спросил, разве не здорово было бы, если бы мы могли осуществлять поиск внутри файлов, я ответил, что мне нужно подумать до завтра. А утром принес эту программу, которая оказалась ровно тем, что он хотел.
>
> С этого момента утилита grep стала и существительным, и глаголом и даже попала в Оксфордский английский словарь. Сложнее всего было придумать ей название. Изначально я назвал ее s (первая буква слова «search» — поиск).

В итоге основой для названия послужила команда `g/re/p`, которая в текстовом редакторе `ed` выводит все строки, соответствующие шаблону, заданному регулярным выражением `re`. Все это изложено в соответствующей статье Оксфордского английского словаря (рис. 4.2). И раз уж понятие grep попало в словарь как легитимное английское слово, в дальнейшем я не буду никак его выделять.



grep, *n.*

Text size: A A

View as: Outline | Full entry Quotations: Show all | Hide all Keywords: On | Off

Pronunciation: Brit. /grɛp/, U.S. /grɛp/

Frequency (in current use):

Origin: Formed within English, by conversion. **Etymon:** English *grep*.

Etymology: < *grep*, a string of characters used as a command in the Unix operating system < the initial letters of *g*lobal(ly) search *r*egular *e*xpression *p*rint.

The string *g/re/p* (where *re* stands for the *r*egular *e*xpression searched for) was earlier used in a Unix text editor as the syntax for a sequence of commands performing the same operation as *grep*.

(Show Less)

Computing.

A Unix command used to search files for the occurrence of a string of characters that matches a specified sequence or pattern, and to output all the lines matching this. Also ***grep command***.

Categories »

1973 *V4/man/man1/grep.1* in *Unix Version 4* (Electronic text) *Grep* will search the input file (standard input default) for each line containing the regular expression. Normally, each line found is printed.

Рис. 4.2. Статья о grep в Оксфордском английском словаре

Моя любимая история, связанная с этой утилитой, случилась в 1972 году. Один из сотрудников Bell Labs заметил, что если перевернуть карманный калькулятор вверх ногами, некоторые выводимые на дисплее цифры становятся похожи на буквы. Например, 3 превращается в E, 7 становится L (рис. 4.3). Так как на нашем компьютере был словарь, нас попросили составить список слов, которые можно набрать на перевернутом калькуляторе.

Рис. 4.3. Буквы BEHILOS на калькуляторе

Как члену подразделения исследований мне было приятно помочь в решении практической задачи. Я спросил, какие буквы можно получить из перевернутых цифр. Оказалось, что это буквы BEHILOS. Тогда я набрал команду

```
grep '^[behilos]*$' /usr/dict/web2
```

Файл `/usr/dict/web2` содержит второе издание международного словаря Уэбстера, то есть 234 936 слов, по одному на строку. Заключенная в кавычки строка символов — регулярное выражение или шаблон, который задает строки, состоящие из произвольных комбинаций семи указанных букв.

Я получил список из 263 слов, представленный на рис. 4.4. Английский — мой родной язык, но в этом списке немало слов, которых я никогда раньше не видел. Я распечатал список и отдал коллеге. Думаю, результат его

удовлетворил, потому что дальнейших вопросов и просьб не было. Зато у меня осталась эта история как прекрасная демонстрация ценности таких инструментов, как утилита grep и регулярные выражения.

```
b       be      bebless  beboss   bee      beeish   beelol
bees    bel     belee    belibel  belie    bell     belle
bes     besee   beshell  besoil   bib      bibb     bibble
bibi    bibless bilbie   bilbo    bile     bilio    bill
bilo    bilobe  bilsh    bios     biose    biosis   bis
bleb    blee    bleo     bless    blibe    bliss    blissless
blo     blob    bo       bob      bobbish  bobble   bobo
boho    boil    bole     bolis    boll     bolo     boo
boob    boohoo  bool     boose    bose     bosh     boss
e       ebb     eboe     eel      eelbob   eh       el
elb     ell     elle     els      else     es       ess
h       he      heel     heelless hei      heii     helbeh
hele    helio   heliosis hell     hellhole hellish  hello
heloe   helosis hi       hie      hill     his      hish
hiss    ho      hob      hobbil   hobble   hobo     hoe
hoi     hoise   hole     holeless holl     hollo    hoose
hoosh   hose    hosel    hoseless i        ibis     ibisbill
ie      ihi     ill      illess   illish   io       is
isle    isleless iso     isohel   issei    l        lee
lees    lei     less     lessee   li       libel    libelee
lie     liesh   lile     lill     lis      lish     lisle
liss    lo      lob      lobbish  lobe     lobeless lobo
lobose  loess   loll     loo      loose    loosish  lose
losel   losh    loss     lossless o        obe      obese
obi     oboe    obol     obole    obsess   oe       oes
oh      ohelo   oho      oii      oil      oilhole  oilless
oleo    oleose  olio     os       ose      osse     s
se      see     seel     seesee   seise    sele     sell
sellie  sess    sessile  sh       she      shee     shell
shi     shiel   shies    shih     shill    shilloo  shish
sho     shoe    shoebill shoeless shole    shoo     shooi
shool   si      sib      sie      sil      sile     sill
silo    siol    sis      sise     sisel    sish     sisi
siss    sissoo  slee     slish    slob     sloe     sloo
sloosh  slosh   so       sob      soboles  soe      soh
soho    soil    soilless sol      sole     soleil   soleless
soles   soli    solio    solo     sool     sooloos  sosh
soso    sosoish soss     sossle
```

Рис. 4.4. Эти слова можно напечатать на перевернутом калькуляторе

Со временем слово grep стало существительным, глаголом и даже герундием (grepping) и вошло в речь Unix-сообщества. Вы когда-нибудь грепали в квартире ключи от машины? Переделали даже слоган AT&T, предлагающий протянуть руку и дотронуться до кого-нибудь (reach out and touch someone).

Протяни руку и грепни кого-нибудь.

Как-то мне позвонил начальник тремя уровнями выше моего Арно Пензиас, лауреат Нобелевской премии и вице-президент отдела исследований, и спросил, допустимо ли использовать эту фразу в публичном выступлении.

4.6. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Выше я использовал словосочетание «регулярное выражение», не объяснив толком, что это такое. Это нотация для задания текстовых шаблонов. Шаблон может выглядеть как слово, например *выражение*, как фраза, например *регулярное выражение*, или как что-то более сложное. По сути, это язык для описания текстовых шаблонов. В обычном языке слово или фраза — это тоже регулярные выражения, и инструмент, который умеет их распознавать, сможет находить все их вхождения.

Сложные шаблоны в регулярных выражениях задаются благодаря так называемым метасимволам, имеющим специальные значения. Например, в grep метасимвол `.` (точка) означает один произвольный символ, а метасимвол `*` (звездочка) задает произвольное число повторений предыдущего символа. Соответственно, под шаблон `(. *)` подходит любая последовательность символов, заключенная в скобки.

Можно сказать, что у Unix с регулярными выражениями давний роман. Ведь они широко применяются в текстовых редакторах, в утилите grep и ее производных и во многих других языках и инструментах. Слегка модифицированные регулярные выражения фигурируют в шаблонах имен файлов, достаточно вспомнить символы подстановки в командной оболочке.

Регулярные выражения использовались в версии текстового редактора QED, которую Кен Томпсон написал для Multics. Именно там я встретил их в первый раз. Кен изобрел быстрый алгоритм обработки выражений произвольной сложности, который даже был запатентован. Редактор QED оказался настолько мощным, что на базе его команд можно было писать программы (хотя никому в здравом уме такое не приходило в голову). Я даже сочинил учебник по программированию в QED, но оказалось, что это усилия, потраченные впустую. Впрочем, они подтолкнули меня к написанию подобной документации.

Большая часть задач, с которыми мы сталкивалась, не требовала такого мощного инструмента, как QED. Соответственно, текстовый редактор Unix `ed`, первую версию которого написали Кен и Деннис, а другие (в том числе я) впоследствии принимали участие в его модификации, был намного проще. Но регулярные выражения использовались и в нем. А так как команда `grep` ведет свое начало от редактора `ed`, она понимает все те же регулярные выражения.

Немного другие регулярные выражения фигурируют в именах файлов в качестве подстановочных символов. Сейчас подстановочные символы интерпретируются оболочкой, но так как оперативная память компьютера PDP-7 была сильно ограничена, в первой реализации для этого применялась отдельная программа `glob` (от слова global — общий, глобальный), вызываемая оболочкой. Акт генерации расширенного списка имен файлов, соответствующих шаблону, называли globbing. Название glob до сих пор присутствует в библиотеках некоторых языков программирования, включая Python.

Одним из ранних вкладов Аль Ахо в Unix стало расширение утилиты grep, позволившее использовать более широкий класс регулярных выражений. Например, появилась возможность указывать альтернативный шаблон `вариант 1|вариант 2`. В этом случае ищутся цепочки символов, удовлетворяющие первому или второму варианту. Свою программу Аль назвал egrep, то есть extended grep (расширенный grep).

Об этой программе имеет смысл рассказать поподробнее, потому что ее история отлично демонстрирует связь между теорией и практикой, а также типичные взаимодействия членов подразделения 1127, которые привели к созданию очень хорошего программного обеспечения. Все это мне рассказал Дуг Макилрой:

Первый предложенный Алем Ахо вариант egrep был обычной реализацией алгоритма из книги *The Design and Analysis of Computer Algorithms*, которую в соавторстве писали Ахо, Хопкрофт и Ульман. Я оперативно добавил его в программу-календарь, в которой для распознавания шаблонов дат, таких как «сегодня», «завтра», «до следующего рабочего дня» и т. д., применялось огромное автоматически сгенерированное регулярное выражение.

К досаде Аля, понадобилось около 30 секунд, чтобы скомпилировать этот алгоритм в распознаватель, который в мгновение ока выдавал результат.

Тогда он придумал блестящую стратегию генерировать распознаватель по мере того, как в нем возникала необходимость. Это позволяло конструировать только маленькую часть из огромного числа состояний. В результате утилита egrep всегда работала быстро, с каким бы сложным шаблоном ей ни приходилось иметь дело. С технической точки зрения это выдающееся решение, но это сложно оценить, если не знать, как плохо работали стандартные методы.

Для Unix это обычная цепочка действий: реальная практическая задача, глубокие знания соответствующей теории, эффективное проектирование, воплощающее теорию в практику, и постоянное усовершенствование. Все собралось воедино благодаря богатому опыту членов группы, открытой среде и культуре экспериментов с новыми идеями.

4.7. ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

С самого начала операционная система Unix была связана с новыми языками программирования.

Проект Multics стал одной из первых попыток написать операционную систему на языке высокого уровня. Для этого выбрали язык PL/I, созданный IBM в 1964 году, как попытку объединить все хорошие идеи из языков Фортран, Кобол и Алгол. Результат может служить замечательным примером эффекта второй системы. Язык получился слишком громоздким и сложным для большинства программистов, при его компиляции возникали трудности, для Multics работающий компилятор так и не был собран. В качестве временной меры Дуг Макилрой и Дуг Иствуд создали упрощенный диалект EPL (Early PL) специально для Multics, но даже он оказался слишком сложным.

Еще одним языком, предназначенным для системного программирования, был BCPL (Basic Combined Programming Language). Его разрабатывал профессор Кембриджского университета Мартин Ричардс, который во время визита в MIT в 1967 году написал для этого языка компилятор. Язык BCPL был намного проще любого диалекта PL/I и хорошо подходил для кода операционной системы. Его хорошо знали члены команды, которая занималась Multics.

После прекращения проекта Multics Кен Томпсон решил, что «без Фортрана не обходится ни один компьютер», и начал писать компилятор этого языка для PDP-7. Задача оказалась чрезмерно сложной, так как под пользовательские программы, такие как компилятор, Unix на PDP-7 могла выделить всего 4 K 18-битных слов (8 КБ).

Кен продолжал искать новые решения и в конце концов придумал язык, подходящий для PDP-7. Он оказался ближе к BCPL, чем к Фортрану. Кен назвал новый язык B. Как объяснил в 1993 году в статье *The Development of the C Language* Деннис Ритчи:

> Язык B можно представить как Си без типов. Точнее, это BCPL, сжатый до 8 Кбайт памяти и профильтрованный через мозг Томпсона. Его название, скорее всего, представляет собой сокращение от BCPL, хотя есть и альтернативная теория, что оно происходит от языка Bon, который Томпсон разработал для использования в Multics. А язык Bon был назван в честь его жены Бонни или (если верить цитате из энциклопедии, добавленной в руководство) в честь старотибетского культа бон, ритуалы которого включают в себя бормотание магических формул.

Все компьютеры, о которых я рассказывал до этого момента, были с пословной, а не с побайтовой обработкой. То есть они манипулировали информацией порциями, превышающими один байт. IBM 7090 и аналогичные компьютеры, например серии GE, естественным образом обрабатывали данные только кусками по 36 битов (примерно четыре байта). У PDP-7 размер кусков составлял 18 битов (два байта). Такие машины с трудом справлялись с обработкой отдельных байтов и их последовательностей. Программистам приходилось прибегать к библиотечным функциям или какими-то другими способами получать доступ к отдельным байтам.

В отличие от этого, PDP-11 был ориентирован на операции над байтами. Основной единицей оперативной памяти стал 8-битный байт, а не 18- или 36-битные слова, как на более ранних компьютерах, хотя PDP-11 умел манипулировать и блоками информации, такими как 16- и 32-разрядные целые и 16-разрядные адреса.

Язык B хорошо подходил для компьютеров с пословной обработкой, таких как PDP-7, а после перехода на PDP-11 Деннис начал дорабатывать B под новую архитектуру и писать для него компилятор. Новый вариант языка сначала получил название NB, что означало Новый B (New B), а чуть позже он превратился в Си.

В новом языке поддерживались типы данных, с которыми работал PDP-11: одиночные байты, двухбайтовые целые числа и в конечном итоге числа с плавающей точкой размером четыре или восемь байт. Там, где такие языки, как BCPL и B, не делали разницы между указателями (адресами памяти) и целыми числами, Си позволял их разделить, хотя долгие годы программисты опрометчиво пренебрегали этой возможностью.

В Си впервые появилась поддержка арифметических операций над типизированными указателями. Указателем называется переменная, содержащая адрес объекта, то есть его местоположение в оперативной памяти. У этой переменной есть тип, это тип объекта, на который она ссылается. Если указатель направлен на элемент массива, то в языке Си добавление 1 перемещает его к следующему элементу. Небрежное обращение с указателями может легко испортить код, тем не менее при корректном использовании арифметические операции с ними отлично работают.

Через некоторое время стало ясно, что для Unix назрел переход с языка ассемблера на язык более высокого уровня. Очевидным выбором стал Си. В 1973 году Кен трижды пытался писать ядро на Си, но это оказалось слишком сложным, пока Деннис не добавил в язык механизм определения и обработки вложенных структур данных (`struct`). С этого момента в Си появилось достаточно средств для написания кода операционной системы, и Unix превратилась в программу на этом языке. В ядре шестой редакции примерно 9000 строк кода на Си и около 700 строк на языке ассемблера, отвечающих за такие машинно-зависимые операции, как настройка регистров, устройств и отображение в память.

Первым получившим широкое распространение описанием языка Си стала наша с Деннисом книга «Язык программирования Си» (рис. 4.5), увидевшая свет в 1978 году; а через десять лет вышло второе издание.

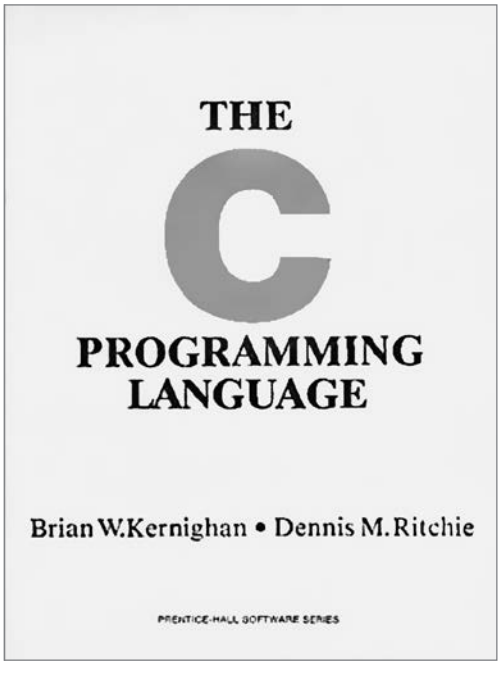


Рис. 4.5. Обложка первого издания нашей книги, 1978 год

Язык B я знал поверхностно, поэтому для собственного удовольствия написал учебник, чтобы помочь себе и другим в его освоении. Когда Деннис создал Си, мне было несложно внести правки в учебник по B и превратить его в учебник по Си. Учебник приобрел популярность, поэтому, наблюдая за распространением Unix и языка Си, я стал задумываться о написании книги. И попросил Денниса стать моим соавтором. Сначала он отказывался, но моя настойчивость взяла вверх. Привлечение Денниса к этой работе было одной из самых больших удач в моей технической карьере. Это придало книге необходимый вес и позволило включить в нее написанный Деннисом справочник.

Мной были написаны черновые варианты большей части учебного материала, а Деннис добавил главу о системных вызовах и, конечно, свой

справочник. Мы много раз совместно редактировали текст, что в итоге сделало стиль книги неоднородным, но справочник остался без изменений. Это целиком стиль Денниса. Он описывает язык с «пугающей точностью», как однажды назвал это Билл Плоджер. Справочник подобен самому языку Си: точный, элегантный и лаконичный.

Первый официальный стандарт языка Си от ANSI, Американского национального института стандартов (а также от ISO, Международной организации стандартов), появился в 1989 году. Вошедшее в стандарт описание языка базировалось непосредственно на справочнике Денниса. Деннис подключился к работе над стандартом Си на ранних стадиях и как создатель этого языка был в состоянии противостоять плохим идеям.

Базовые средства для таких операций, как форматированный ввод и вывод, обработка строк и математические функции, в Си предоставляет стандартная библиотека. Благодаря ей программистам не приходится при написании каждой программы с нуля изобретать все процедуры.

Самый большой компонент библиотеки обеспечивает форматированный вывод. Это знакомая всем современным программистам функция `printf`, которая была адаптирована под множество других языков. Первую версию этой функции содержал портативный пакет ввода-вывода (portable I/O package), написанный Майком Леском в 1972 году, чтобы облегчить перемещение программ в Unix и из нее. Еще этот пакет включал функцию `scanf`, предназначенную для анализа форматированного ввода. Обе эти функции после переработки были включены в компилятор Си.

Несмотря на появившиеся расширения, основной набор преобразований в функциях `printf` и `scanf` работает так же, как и в начале 1970-х годов. Аналогичным образом обстоят дела с большинством других функций в библиотеке. Сейчас стандартная библиотека — такая же часть стандарта Си, как и сама спецификация языка.

Интересно сопоставить подходы из Си с происходящим в других языках. Например, в Фортране и Паскале существует специальный синтаксис для чтения и записи данных, так как операции ввода и вывода — это тоже часть языка. При этом есть и языки, в которые эти операции не включены и которые не предоставляют стандартную библиотеку, что можно считать одной из самых плохих комбинаций.

Из широко используемых языков Си можно считать очень успешным. Операционная система Unix, появившаяся на PDP-11, распространилась практически на все разновидности компьютеров, которые существуют в природе. Как сказал Деннис в своей статье для второй конференции по истории языков программирования 1993 года (*History of Programming Languages*):

> Это самобытный, небезупречный и снискавший огромный успех язык. Пусть этому успеху и содействовали исторические случайности, невозможно отрицать, что при всей своей абстрактности и гибкости, подходящей для описания алгоритмов, он настолько эффективно удовлетворил потребность в языке реализации систем, что смог вытеснить с этого места язык ассемблера.

Конечно, есть множество других языков программирования, приверженцы которых часто критикуют Си. Тем не менее он остается основным языком вычислений и в списках популярности и важности почти всегда попадает в первую тройку. На мой взгляд, ни в одном другом языке не смогли достичь такого баланса элегантности, выразительности, эффективности и простоты. Си также послужил основой синтаксиса многих других языков, в том числе C++, Java, JavaScript, Awk и Go. А это говорит очень о многом.

4.8. КНИГА SOFTWARE TOOLS И ЯЗЫК RATFOR

Ближе к концу 1975 года описания операционной системы Unix фигурировали на конференциях и в журнальных статьях, шестой редакцией пользовались сотни университетов, появилось и некоторое количество коммерческих применений. Но в мире техники по большей части все еще использовали Фортран и работали в операционных системах от поставщиков оборудования, например таких, которые IBM устанавливала на мейнфреймы System/360. У нас в Мюррей-Хилл большинство программистов работало на машинах GE 635, где была установлена система с пакетной обработкой данных GECOS (после того как в 1970 году фирма GE продала свой компьютерный бизнес корпорации Honeywell, ее переименовали в GCOS).

К 1973 году я начал регулярно программировать на Си, несмотря на то что все еще писал на Фортране. Хотя этот язык отлично подходил для научных

и инженерных вычислений, в нем практически не существовало операторов управления потоком выполнения. Кроме того, присутствовали ограничения, обусловленные тем, что этот язык появился в 1950-х, когда структура программ ориентировалась на ввод с перфокарт. В Си же поток выполнения контролировался, если можно так выразиться, естественным образом.

В итоге я написал простой компилятор, который переводил похожий на Си диалект Фортрана в нормальный Фортран. Я назвал его *Ratfor* (сокращение от Rational Fortran — рациональный Фортран). Этот препроцессор преобразовывал поток команд управления Си с его операторами `if-else`, `for`, `while` и скобками для группировки в операторы Фортрана `IF` и `GOTO` и в единственный цикл `DO`. Обеспечивал он и удобства обозначений, например ввод в произвольной форме (а не фиксированный в соответствии с 80-колоночными перфокартами формат, который в то время все еще требовался в Фортране), удобное обозначение комментариев, естественные логические операторы и операторы сравнения, то есть `<` и `>=` вместо громоздких .LT и .GE.

В качестве небольшого примера покажу один из способов, которым программу на Фортране из главы 1 можно написать на языке Ratfor:

```
# создание единичной матрицы v
do i = 1, n
    do j = 1, n
        if (i == j)
            v(i,j) = 1.0
        else
            v(i,j) = 0.0
```

Ratfor стал первым примером языка с синтаксисом на основе Си. Написание программ на Фортране на языке Ratfor было, если можно так выразиться, немного приятнее, чем на стандартном Фортране. Семантику и типы данных Фортрана Ratfor не менял; например, в нем не было функций обработки символов, но там, где подходил Фортран, Ratfor справлялся лучше. Благодаря произвольному формату ввода и напоминающим Си управляющим конструкциям возникало ощущение, что вы пишете на Си.

Виртуозно объединив теорию с практикой, Бренда Бейкер создала программу `struct`, позволяющую перевести любую программу с Фортрана на Ratfor. Бренда показала, что почти все программы на Фортране хорошо

структурированы, а для описания этой структуры лучше всего подходит Ratfor. Те, кто пользовался `struct`, обнаруживали, что предлагаемая версия на Ratfor почти всегда оказывалась понятнее исходной программы на Фортране.

Мы с Биллом Плоджером решили написать книгу, пропагандирующую философию и инструментарий Unix более широкой аудитории, а именно программистам, пишущим на Фортране для других операционных систем. Вышедшая в 1976 году книга *Software Tools* представила версии стандартных инструментов Unix на языке Ratfor. Там были инструменты для сравнения файлов и подсчета слов, grep, редактор, похожий на `ed`, напоминающее roff средство форматирования текстов, кроме того, сам препроцессор Ratfor. Все они были написаны на языке Ratfor.

Со сроками мы угадали. Книга хорошо продавалась, появилась организация Software Tools User Group, возглавляемая сотрудниками Ливерморской национальной лаборатории Дебби Шеррер, Деннисом Холлом и Джо Свентеком. Они дорабатывали и совершенствовали программы, добавляли собственные инструменты, устраивали раздачи кода, организовывали конференции и много лет поддерживали группу. Написанный ими код был портирован на более чем 50 операционных систем. До самого распада в конце 1980-х эта организация оставалась активной и авторитетной.

В 1981 году мы с Биллом выпустили версию *Software Tools* для языка Паскаль, популярного в то время в университетах в качестве учебного языка. Паскаль обладал хорошими свойствами, в том числе разумными управляющими конструкциями и рекурсией, которых так не хватало в Фортране.

К сожалению, у него есть и недостатки, такие как неуклюжая система ввода и вывода и почти неиспользуемые символьные строки. Я описал их в статье «Почему Паскаль не мой любимый язык программирования»[1]. Статья была отдана в журнал, но отклонена как слишком спорная или недостаточно значимая. Она так никогда и не была официально опубликована, хотя на удивление часто цитируется.

В любом случае по мере увеличения доступности Си и Unix популярность языка Паскаль уменьшалась из-за его серьезных ограничений, и книга *Software Tools in Pascal* осталась невостребованной. Сейчас я понимаю,

[1] http://www.lysator.liu.se/c/bwk-on-pascal.html

что как в краткосрочной, так и в долгосрочной перспективе разумнее было писать про версию для языка Си.

4.9. БИОГРАФИЯ: ДУГ МАКИЛРОЙ

Роб Пайк назвал Дуга Макилроя неоцененным героем Unix, и я с ним согласен. По словам Кена Томпсона, Дуг умнее всех, что тоже не вызывает возражений, хотя сам Дуг говорит: «Я предпочитаю, чтобы мой ум оценивали другие, при этом мне знакомы практикующие математики из Bell Labs, которые намного умнее меня». Впрочем, в Bell Labs, где трудилось множество выдающихся людей, синдром самозванца был достаточно распространенным явлением и прилагались большие усилия в попытках соответствовать.

Но как бы в действительности ни обстояли дела, существует вероятность, что без хорошего вкуса и здравого смысла Дуга по отношению как к техническим вопросам, так и к людям операционная система Unix, возможно, не появилась бы и уж точно не стала бы настолько успешной.

Дуг получил степень бакалавра в области инженерной физики в Корнелльском университете в 1954 году и степень доктора наук в области прикладной математики в MIT в 1959 году. У него тоже были летние стажировки в Bell Labs, а окончательно он присоединился к ним в 1958 году. В 1965 году, за два года до нашей встречи, он стал руководителем отдела исследований вычислительной техники (Computing Techniques Research department). Как я уже рассказывал, лето 1967 года я провел в качестве стажера в отделе Дуга, номинально работая над предложенной им задачей распределения памяти, но фактически занимаясь своими делами. Одним из его качеств как руководителя было полное спокойствие по этому поводу.

Выше я упоминал раннюю работу Дуга над языками PL/I и EPL. В процессе разработки Unix Дуг написал множество фундаментальных программ. Его функция выделения памяти `malloc` использовалась в течение многих лет, а исследование механизмов распределения памяти повлияло на последующие разработки. На посвященной ему странице Дартмутского колледжа перечислены команды его авторства `spell`, `diff`, `sort`, `join`, `graph`, `speak`, `tr`, `tsort`, `calendar`, `echo` и `tee`.

В этом списке присутствуют как небольшие команды, например `echo`, так и целые утилиты, например `sort` и `diff`, но большая часть списка играет принципиальную роль в вычислительных процессах Unix. Многие команды используются и по сей день. Разумеется, конвейеры тоже были его идеей, хотя в окончательной версии использовался синтаксис Кена. Но самим своим существованием конвейеры обязаны тому, что Дуг постоянно продвигал этот механизм.

Написанная им версия команды `spell` эффективно использовала словарь и эвристику для определения частей речи, чтобы в условиях недостатка ресурсов искать потенциальные орфографические ошибки.

Предложенная Дугом версия команды `diff` реализует эффективный алгоритм (независимо изобретенный Гарольдом Стоуном и Томом Шиманski) сравнения двух текстовых файлов и нахождения минимальной последовательности изменений для преобразования одного в другой. Этот код лег в основу систем контроля исходного кода, которые управляют версиями файлов. Чаще всего такие системы сохраняют одну версию и набор различий, а генерация версий осуществляется с помощью алгоритма `diff`. Применяется он и в механизме установки исправлений, который используется для обновления программ. Пользователям отправляется не новая версия программы, а рассчитанная с помощью алгоритма `diff` последовательность команд редактирования `ed`, которые преобразуют старую версию в новую.

Утилита `diff` — еще один хороший пример фундаментального инструмента, который создается, когда хорошую теорию сочетают с качественной практической разработкой. Вывод этой утилиты читабелен как для людей, так и для программ. Вывод, написанный специально для человека или для программы, оказался бы куда менее полезным. Фактически мы видим программу, которая осуществляет вывод в программу, и это отдельный пример небольшого компьютерного языка.

Вскоре после появления Unix в подразделении 1127 приобрели новое устройство — голосовой синтезатор Votrax, преобразовывающий фонемы в звуки. Дуг разработал набор правил преобразования произвольного английского текста в фонемы и написал программу `speak`, которая на их основе генерировала ввод для синтезатора Votrax. Конечно, английская орфография печально известна многочисленными исключениями из правил, так что до идеала была далеко. Результат работы программы `speak` часто

оказывался несовершенным, а порой и очень забавным (например, мое имя зарифмовали Br-I-an Kern-I-an), но почти всегда достаточно точным, чтобы его можно было использовать.

Программа `speak` была, по сути, командой, и ею мог воспользоваться кто угодно. Отправленный в нее текст воспроизводился в комнате Unix через динамик. Это породило множество дополнительных объявлений. Например, каждый день в 13:00 Votrax говорил:

```
Время обеда, время обеда, время обеда. Ням, ням, ням,
```

напоминая обитателям комнаты, что неплохо бы выйти в кафе до того, как в 13:15 оно закроется.

Появилась также служба контроля входящих телефонных звонков. В момент звонка в тишине раздавалось объявление:

```
Звонок для Дуга.
```

Или для другого человека. В общей комнате это отвлекало намного меньше, чем постоянно звонящие телефоны.

Интересы Дуга были крайне разнообразны. Среди прочего он был экспертом в математической картографии, то есть в способах построения картографических проекций. Написанная им программа `map` сгенерировала несколько десятков проекций (рис. 4.6). Некоторые из них можно увидеть на странице Дуга на сайте Дартмутского колледжа.

Дуг часто оказывался первым, кто пробовал какую-то новую программу или идею. Обнаружив новинку, он сразу начинал с ней экспериментировать, а так как у него был отличный вкус, его мнение о достоинствах и недостатках высоко ценилось. Я помню постоянный поток посетителей Дуга, которые шли получить совет и критические замечания к своим идеям, алгоритмам, программам, документам — почти ко всему. Бывало, Бьёрн Страуструп заходил ко мне обсудить C++ и объяснить какую-то новую идею, а затем проходил дальше по коридору в кабинет Дуга, чтобы получить замечания о проектировании языка.

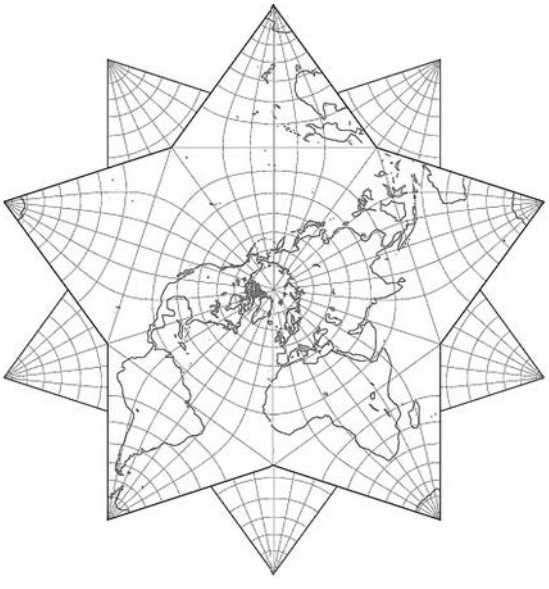


Рис. 4.6. Одна из множества картографических проекций Дуга Макилроя

Обычно Дуг первым читал черновики документов или руководств и ловко убирал избыточную лексику, расплывчатый текст, ненужные наречия, словом, наводил порядок. Вот что рассказал о Дуге Аль Ахо в рамках проекта Майка Махони *Oral history of Unix* (1989):

> Он даже по самому отрывочному описанию мог понять все, над чем я работал. По сути, это он научил меня писать. Я думаю, что это один из лучших технических писателей, которых я знаю. У него есть чутье языка и замечательный талант лаконично выражать свои мысли.

Дуг читал мою диссертацию и помог улучшить как ее структуру, так и содержание. Прочел он и несколько черновиков всех книг, написанных мной в соавторстве с коллегами по Bell Labs, что всегда позволяло сделать их лучше. Он довел до совершенства страницы справочника команд, а также собрал и систематизировал содержимое справочников по 8–10-й редакциям Unix. Все это он делал с энтузиазмом и вниманием, что порой сказывалось на его собственных исследованиях.

В 1986 году Дуг уволился с руководящей должности, а в 1997 году перешел из Bell Labs в Дартмут, где занялся преподаванием. Фото 4.7 сделано в Мюррей-Хилл в 2011 году во время вручения Премии Японии Кену и Деннису.

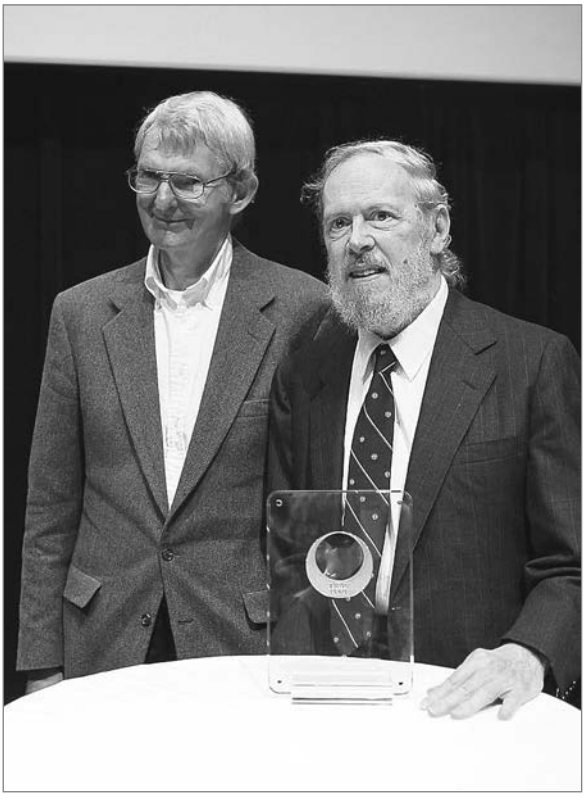


Рис. 4.7. Дуг Макилрой и Деннис Ритчи, май 2011 года (Википедия)

ГЛАВА 5

СЕДЬМАЯ РЕДАКЦИЯ (1976–1979)

> На самом деле только к седьмой редакции система оперилась и покинула родное гнездо. Это была первая переносимая версия, последний общий родоначальник всех бесчисленных реализаций системы для разных аппаратных платформ. Поэтому история седьмой редакции — часть общего наследия всех систем семейства Unix.
>
> *Дуг Макилрой, статья A Research Unix Reader, 1986*

Шестая редакция Unix стала отличной базой для разработки программного обеспечения, а поставляемые вместе с ней инструменты сделали программирование увлекательным и продуктивным. Некоторые фрагменты системы появились задолго до шестой редакции, а что-то было добавлено позже. В этой главе я расскажу про различные потоки разработки ПО в подразделении 1127, кульминацией которых стала седьмая редакция, вышедшая в январе 1979 года, почти через четыре года после шестой.

Логически и хронологически некоторые материалы должны были следовать после главы, описывающей выход Unix за пределы подразделения 1127, но история примет более целостный вид, если сначала описать седьмую редакцию. Как отметил Дуг Макилрой в статье, фрагмент кото-

рой вынесен в эпиграф этой главы, именно эта редакция стала источником большей части того, что было унаследовано всеми системами семейства Unix.

В этой главе я буду много говорить на такую важную тему, как расцвет языков. Некоторые из языков предназначались для традиционного программирования, часть была определена под конкретную предметную область, а часть требовалась для написания спецификаций. Подозреваю, что мой рассказ получится намного более подробным, чем интересно большинству читателей, но это тема, которой я занимался много лет. Постараюсь самую важную часть информации давать вначале, чтобы можно было спокойно переходить к следующему разделу.

Следует отметить, что если шестая редакция Unix предназначалась исключительно для PDP-11, то появившаяся к 1979 году седьмая редакция представляла собой кроссплатформенную операционную систему, работающую как минимум на четырех архитектурах, из которых самой популярной была VAX-11/780 от DEC. О портируемости мы подробно поговорим в следующей главе, а сейчас важно только то, что из системы для PDP-11 Unix незаметно превратилась в систему, сравнительно независимую от конкретного оборудования.

5.1. ОБОЛОЧКА БОРНА

Добавление к командной оболочке в шестой редакции системы перенаправления ввода/вывода и конвейеров позволили легко комбинировать программы. Достаточно было создать файл с последовательностью команд — сценарий оболочки — и запускать всю последовательность одной командой.

В шестой редакции оболочка понимала такие управляющие конструкции, как условный оператор `if` и оператор перехода к другой строке файла сценария `goto`. Кроме того, был способ помечать строки сценария (команда `:`, которая ничего не делала). Все это позволяло реализовывать ветвления и даже циклы — словом, в шестой редакции оболочка уже подходила для написания сложных сценариев. Однако на практике все эти механизмы были неудобными и неустойчивыми.

В следующей главе я расскажу, как Джон Мэши — член группы, занимавшейся разработкой дистрибутива Programmer's Workbench (PWB), — добавил к своей версии оболочки ряд функций, упростивших процесс программирования: общий оператор `if-then-else` для проверки условий, оператор `while` для циклов и переменные для хранения информации внутри оболочки.

В 1976 году недавно присоединившийся к подразделению 1127 Стив Борн написал новую оболочку, добавив к функциям PWB-оболочки ряд усовершенствований. У него была цель сохранить работу в интерактивном режиме, добавив возможность написания сценариев. Новая оболочка Стива поддерживала такие конструкции для управления потоком выполнения, как `if-then-else`, `while`, `for` и `case`, а также переменные окружения, определяемые как самой оболочкой, так и пользователями. Были улучшены механизмы квотирования. Наконец, появилась возможность встраивать команды оболочки в конвейер, как и любую другую программу. Эта версия получила название `sh` и быстро вытеснила оболочку шестой редакции.

Новая оболочка обладала необычным синтаксисом управления потоком выполнения, поскольку базировалась на языке Algol 68, предпочитаемом Стивом, но непопулярном среди остальных сотрудников подразделения 1127. Например, в качестве закрывающих ключевых слов в Algol 68 использовались их перевернутые версии, например `fi` для завершения `if` и `esac` для завершения `case`. А так как вариант `od` уже применялся для получения восьмеричного дампа, для завершения оператора `do` использовалось ключевое слово `done`.

```
for i in $* перебираем в цикле все аргументы
do
    if grep something $i
    then
        echo found something in $i
    else
        echo something not found in $i
    fi
done
```

Условные операторы `if` и `while` возвращают статус завершения в виде числового значения, которое показывает, как программа закончила свою работу. Правда, в то время авторы большинства программ не особо щепе-

тильно подходили к вопросу возвращения значимых статусов, так как это не имело особого значения. Поэтому Стив настроил оболочку так, чтобы каждый раз, когда программа возвращала нечто непонятное, появлялось раздраженное сообщение. Через неделю такого автоматизированного ворчания большинство программ были обновлены и начали возвращать значимые статусы.

В своей версии оболочки Стив значительно усовершенствовал перенаправление ввода/вывода. В разделе BUGS про оболочку шестой редакции говорилось: «Невозможно перенаправить вывод результатов диагностики». Одна из полезных функций новой оболочки — разделение стандартного потока ошибок (по умолчанию файловый дескриптор 2) и стандартного вывода (файловый дескриптор 1), позволяющее направить результаты работы сценария в файл, а сообщения об ошибках — в другое место, обычно на терминал:

```
prog >file            # stdout в файл, stderr на терминал
prog 2>err            # stdout на терминал, stderr в файл err
prog 1>file 2>err     # stdout в файл, stderr в файл err
prog >file 2>&1       # слияние stderr и stdout
```

К этому моменту оболочка превратилась в самостоятельный язык программирования, подходящий для написания практически всего, что можно представить в виде последовательности команд. Часто результат оказывался настолько хорош, что отпадала необходимость писать программу на Си.

За прошедшие годы в командные интерпретаторы была добавлена дополнительная функциональность, особенно в Bash, который сейчас фактически является стандартной оболочкой для пользователей Linux и macOS. Название расшифровывается как Bourne Again Shell (еще одна командная оболочка Борна) и перекликается с выражением «Born again shell» — «возрожденный shell». Пользовательские сценарии для оболочки, как правило, небольшие и простые, а вот исходный код основных инструментов, таких как компиляторы, часто распространяется с конфигурационными сценариями, насчитывающими 20 000 строк или более. Такие сценарии запускают программы проверки свойств среды, например наличия библиотек и размеров типов данных, что позволяет им компилировать версию, настроенную под конкретную систему.

5.2. YACC, LEX, MAKE

Для общения люди используют язык, и более развитые языки помогают общаться более эффективно. Это особенно верно для искусственных языков, предназначенных для общения с компьютерами. Хороший язык уменьшает разницу между тем, что мы хотим сказать («сделай это»), и тем, что мы должны сказать, чтобы получить нужный результат. Созданию выразительных языков посвящено большое количество исследований в области вычислительной техники.

Седьмая редакция Unix предложила множество языковых инструментов, среди которых встречаются и нетрадиционные. Справедливо заметить, что большинства этих языков не появилось бы, не будь инструментов, позволяющих неспециалистам создавать новые языки, особенно программы Yacc. В этом разделе мы поговорим о таких инструментах. Но я думаю, что главную идею вы уже поняли. Инструменты Unix способствовали созданию новых языков, тем самым улучшая способы общения с компьютерами. Собственно, это и есть самая важная информация. Подробности, изложенные ниже, вы можете спокойно пропустить.

Два основных аспекта, характеризующих компьютерные языки, это синтаксис и семантика. Синтаксис описывает грамматику: то, как выглядит язык, что в нем допустимо, а что нет. Он задает правила написания операторов и функций, определяя, что такое арифметические и логические операторы, как они объединяются в выражения, какие имена допустимы, какие слова зарезервированы, как выражаются строковые и числовые литералы, как форматируются программы и т. д.

Семантика формализует значения, приписываемые допустимому синтаксису. Именно она определяет, что означает или делает допустимая конструкция. Рассмотрим программу вычисления площади из главы 2:

```
void main() {
    float length, width, area;
    scanf("%f %f", &length, &width);
    area = length * width;
    printf("площадь = %f\n", area);
}
```

Семантически эта программа читается так: вызывается функция `main`, которая вызывает функцию `scanf` для считывания двух значений из стан-

дартного ввода, вычисления площади и вызова функции `printf`, которая выведет строку `площадь =`, значение переменной `area` и символ перевода строки (`\ n`).

Компилятором называется программа, которая переводит написанное на одном языке в семантический эквивалент на другом языке. Например, компиляторы для языков высокого уровня, таких как Си и Фортран, могут выполнять перевод на язык ассемблера для определенного типа компьютера. Некоторые компиляторы переводят на Фортран с других языков, таких как Ratfor.

Первая часть процесса компиляции — это *анализ* программы, то есть разбор ее синтаксической структуры путем распознавания имен, констант, определений функций, порядка выполнения, выражений и тому подобного, что позволит при последующей обработке присоединить подходящую семантику.

Сегодня написание синтаксических анализаторов для языков программирования — хорошо разработанная технология, но в начале 1970-х годов велись активные исследования, нацеленные на создание программ, способных преобразовать грамматические правила языка в эффективный синтаксический анализатор для написанного на этом языке кода. Такие программы называли компилятором компиляторов, поскольку они позволяли механически генерировать синтаксический анализатор для компилятора. Как правило, они создают анализатор (который еще называют парсером), заодно предоставляя способ выполнения кода, который может встретиться во время синтаксического анализа.

YACC

В 1973 году Стив Джонсон (рис. 5.1) с помощью советов по теории языка от Аля Ахо создал компилятор компиляторов, который назвал YACC (далее Yacc). В основе названия — комментарии Джеффа Уллмана (Yet Another Compiler Compiler — еще один компилятор компиляторов), который дает понять, что это была не первая подобная программа.

Программа Yacc состоит из грамматических правил для синтаксиса языка и семантических действий, связанных с этими правилами, чтобы при обнаружении во время анализа конкретной грамматической конструкции можно было выполнить соответствующее семантическое действие.

Например, в виде псевдокода часть синтаксиса арифметических выражений может выглядеть так:

```
выражение := выражение + выражение
выражение := выражение * выражение
```

Семантические действия в этих случаях сведутся к генерации кода, который будет складывать или умножать результаты двух выражений и выполнять присваивание полученного результата. Программа Yacc преобразует такую спецификацию в программу на Си, которая анализирует ввод и в процессе этого анализа выполняет семантические действия.

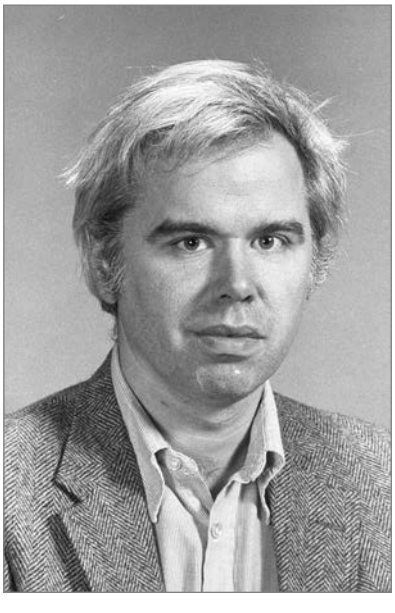


Рис. 5.1. Стив Джонсон, примерно 1981 год (из архива Джерарда Хольцманна)

Обычно разработчику компилятора приходится писать более сложные правила, так как следует учитывать приоритет операции умножения (умножение выполняется перед сложением), но в Yacc приоритеты операторов и ассоциативность можно задать отдельными объявлениями, а не дополнительными грамматическими правилами, что сильно упрощает ситуацию для неопытных пользователей.

Сам Стив использовал Yacc для создания нового портативного компилятора Си (portable C compiler, PCC), который имел общий интерфейс для синтаксического анализа языка и отдельные модули для генерации кода под различные компьютерные архитектуры. Стив и Деннис также использовали PCC при реализации Unix для компьютера Interdata 8/32, о которой пойдет речь в разделе 6.5.

Были у PCC и другие варианты применения. Вот что вспоминал Стив:

> Неожиданным побочным продуктом PCC стал статический анализатор Lint. Он читал программу и сообщал о непереносимостях на другие платформы или ошибках, например вызов функции с неправильным количеством аргументов, несовместимость размеров между определением и применением и т. п. Компилятор Си просматривал файлы по очереди, поэтому Lint быстро стал полезным инструментом при написании многофайловых программ. Помогал он и следить за соблюдением стандартов, когда мы работали над портируемостью седьмой редакции. Именно он искал, например, системные вызовы, которые возвращали код ошибки –1 (редакция 6) вместо нуля (редакция 7). Многие такие проверки, даже проверки портируемости, в конечном итоге были введены в сам язык Си; Lint оказался полезным тестовым стендом для новой функциональности.

Название *Lint* восходит к слову, означающему снятие ворсинок с одежды. Функциональность этого анализатора теперь часто включается в компиляторы Си, но его базовая идея легла в основу аналогичных инструментов для ряда других языков.

Анализатор Yacc сыграл важную роль в создании некоторых языков, которые разрабатывались в подразделении 1127. Про некоторые из них я расскажу в следующих разделах. Вместе с Лориндой Черри мы использовали его для языка описания математических выражений Eqn. Я также использовал Yacc для препроцессоров Pic и Grap (последний был сделан в соавторстве с Джоном Бентли, рис. 5.2), для языка программирования AMPL, как минимум для одной версии Ratfor и для других языков. Применялся Yacc и для f77 (первого компилятора языка Fortran 77, для препроцессора C++ авторства Бьёрна Страуструпа), языка сценариев awk (о нем мы поговорим чуть позже) и множества других целей.

Рис. 5.2. Джон Бентли, примерно 1981 год
(из архива Джерарда Хольцманна)

Сочетание продуманной технологии синтаксического анализа, высокой эффективности и удобного пользовательского интерфейса помогло программе Yacc остаться единственным выжившим из первых программ генерации синтаксических анализаторов. Сегодня эта программа существует как под собственным именем, так и в независимых реализациях, таких как Bison, и в переизданиях на другие языки.

LEX

В 1975 году Майкл Леск (рис. 5.3) создал генератор лексического анализатора Lex, который является прямой параллелью Yacc. Программа Lex состоит из последовательности шаблонов (регулярных выражений), определяющих подлежащие идентификации лексические токены. Для языка программирования это будут такие компоненты, как зарезервированные слова, имена переменных, операторы, знаки пунктуации и т. д. Как и в случае с Yacc, к любому токену можно присоединить семантическое действие, написанное на Си. Из совокупности таких токенов Lex генерирует программу на Си, умеющую считывать поток символов, идентифицировать обнаруженные токены и выполнять соответствующие семантические действия.

Рис. 5.3. Майкл Леск, примерно 1981 год (из архива Джерарда Хольцманна)

Майк написал первую версию Lex, но уже летом 1976 года она была пересмотрена стажером, который только что окончил Принстон. Майк вспоминает:

> Программа Lex была почти сразу переработана пришедшим на летнюю стажировку Эриком Шмидтом. Я написал ее с помощью недетерминированного анализатора, который не мог обрабатывать правила с более чем 16 состояниями. Аль Ахо расстроился и дал мне стажера, чтобы исправить ситуацию. А стажер оказался выдающимся человеком.

Эрик защитил докторскую диссертацию в Беркли, а с 2001 по 2011 год работал главным исполнительным директором в Google.

Программы Yacc и Lex хорошо работают вместе. Каждый раз, когда Yacc в процессе анализа требует следующий токен, он вызывает Lex, который считывает достаточное количество входных данных для идентификации полного токена и передает его в Yacc. Комбинация Yacc/Lex реализует внешние компоненты компилятора, одновременно обрабатывая сложные грамматические и лексические конструкции. Например, в некоторых языках программирования есть операторы длиной два или даже три символа. Скажем, оператор ++ в Си. Когда лексический анализатор видит символ +, ему нужно узнать следующий символ, чтобы понять, какой перед ним опе-

ратор: ++ или обычный +. Написать такой код вручную не так уж сложно, но еще лучше, когда это делается за вас. В программе Lex для разграничения этих двух случаев достаточно написать:

```
"++" { return PLUSPLUS; }
"+" { return PLUS; }
```

Здесь PLUS и PLUSPLUS — это имена для численных кодов, с которыми легко иметь дело в коде на языке Си.

На рис. 5.4 показана схема применения программ Yacc и Lex для создания программы на Си, которая является компилятором какого-то языка. Yacc создает файл Си для синтаксического анализатора, а Lex — файл Си для лексического анализатора. Эти два файла объединяются с другими файлами Си, которые содержат семантику, а затем компилятор Си генерирует исполняемую программу. Фигура на рисунке получена с помощью препроцессора Pic, предназначенного для построения диаграмм.

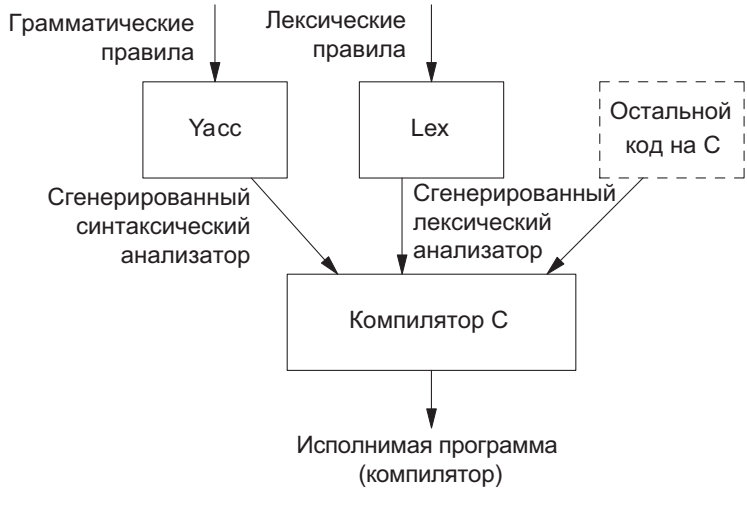


Рис. 5.4. Создание компилятора с помощью программ Yacc и Lex

Несмотря на сильные стороны программы Lex, долгое время она не применялась настолько же широко, как Yacc. Возможно, это связано с тем, что неопытному программисту трудно написать синтаксический анализатор для сложных языков, а с лексическим анализатором он вполне справится. Другое дело, что это далеко не всегда хорошая идея, каким

бы простым и понятным ни казался процесс написания лексического анализатора.

Поучителен мой опыт работы с языком сценариев Awk. Первая реализация этого языка использовала Yacc для анализа грамматики, а Lex — для разбивки входной программы на токены. Потом мы попытались перенести Awk в другие среды и столкнулись с тем, что программы Lex там не было или она работала по-другому. Через несколько лет я с неохотой написал лексическую часть Awk на Си, обеспечив его переносимость в другие среды. И много лет этот написанный вручную код был богатым источником ошибок и неочевидных проблем, с которыми мы не сталкивались в сгенерированной Lex версии.

Это хороший пример общего правила: сгенерированный программой код оказывается более правильным и надежным, чем написанный вручную. От совершенствования генератора выигрывают все, в то время как совершенствование одной рукописной программы никак не влияет на другие. Такие инструменты, как Yacc и Lex, отлично иллюстрируют это правило, а операционная система Unix дает множество других примеров. Всегда лучше пытаться писать программы, которые пишут программы. Как говорит Дуг Макилрой, «все, что вам придется делать неоднократно, можно считать готовым к автоматизации».

MAKE

Большинство крупных программ состоит из набора исходных файлов, которые требуется скомпилировать и объединить в исполняемый файл. Часто это можно сделать одной командой. Например, команда `cc * .c` осуществляет компиляцию всех исходных файлов для программы на Си. Но в 1970-х компьютеры работали так медленно, что перекомпиляция многофайловой программы после внесения изменений в один файл могла занять значительное время. Рациональнее было перекомпилировать только измененный файл и связать результат с ранее скомпилированными файлами.

Правда, при таком подходе требовалось помнить, какие файлы недавно компилировались, а какие нужно перекомпилировать сейчас, и ошибиться было очень легко. Однажды Стив Джонсон пожаловался на это Стю Фельдману (рис. 5.5) после того, как после многочасовой безуспешной отладки понял, что просто забыл перекомпилировать один из измененных файлов.

Рис. 5.5. Стю Фельдман, примерно 1981 год (из архива Джерарда Хольцманна)

По случайному совпадению в это время Стю тоже пытался отладить корректную, но не перекомпилированную программу. Ему пришла в голову элегантная идея: язык спецификаций, описывающий зависимость частей программы друг от друга. Программа, которую он назвал Make, анализировала спецификацию и, базируясь на времени изменения каждого файла, выполняла минимальную перекомпиляцию, необходимую для обновления всего пакета. Первая версия программы появилась в 1976 году:

> Программу Make я написал за выходные, а на следующих выходных переписал ее с помощью макросов (потому что иначе список встроенного кода становился слишком длинным). Начало командных строк с символа табуляции[1] я оставил, потому что у меня быстро набралась база из более чем дюжины человек, и я не хотел их расстраивать.

[1] В программе Make строки, в которых записаны команды, должны начинаться с символа табуляции. Если вместо этого символа поставить набор пробелов, сценарий работать не будет, и эту ошибку очень трудно обнаружить. Фельдман не стал исправлять этот момент, чтобы сохранить обратную совместимость для самых первых пользователей. — *Примеч. пер.*

Программа Make имела успех, поскольку избавляла от целого класса глупых ошибок, тем самым обеспечивая максимальную эффективность компиляции. Принесла она пользу и программам, которые производили более сложную обработку. Например, программам, которые, как в диаграмме с рис. 5.4, сначала создавали файлы на Си с помощью генераторов Yacc и Lex, а только потом компилировались. Один make-файл может не только охватить все этапы, необходимые для компиляции новой версии программы, но и описать способ выполнения связанных задач, таких как запуск анализатора Lint, создание резервной копии и вывод документации. Некоторые свойства make-файла аналогичны свойствам сценария оболочки, но для них использовался декларативный язык, который определял спецификацию зависимостей и способы обновления компонентов, но не занимался явной проверкой времени создания файлов.

Проблема символа табуляции в начале командных строк — нестандартное и труднопреодолимое ограничение, наложенное на формат make-файлов. Можно утверждать, что это ошибка проектирования. Но этот пример хорошо иллюстрирует трудности, с которыми сталкивается любая успешная программа. Она быстро привлекает пользователей, что затрудняет внесение коренных изменений в ее структуру. Как в Unix, так и в других операционных системах можно вспомнить множество примеров первоначальных недоработок, которые слишком укоренились, чтобы их можно было исправить.

В то же время программа Make служит прекрасной иллюстрацией темы этого раздела. Вместо того чтобы вручную писать код или выполнять последовательности операций, лучше создать нотацию или спецификацию того, что нужно сделать, а потом написать программу для интерпретации этой спецификации. Такая замена кода данными почти всегда дает выигрыш.

Программы Yacc, Lex и Make активно используются и сегодня, потому что решают важные проблемы, с которыми все еще сталкиваются программисты, и решают их достаточно хорошо, чтобы их продолжали применять, несмотря на особенности проектирования, а иногда и оригинальные реализации.

В заключение расскажу, что со Стю мы впервые встретились в 1967 году, когда я был аспирантом в Принстоне, а он еще студентом и в течение

учебного года работал по совместительству над проектом Multics для Bell Labs. К подразделению 1127 он присоединился после получения докторской степени по астрофизике в MIT. В 1984 году он перешел в компанию Bellcore, затем в IBM, затем в Google, где, к счастью для меня, во время моих летних визитов туда я оказывался под его руководством.

5.3. ПОДГОТОВКА ДОКУМЕНТОВ

В Unix изначально имелись прекрасные инструменты для подготовки документов, что способствовало появлению хорошей документации. В этом разделе я расскажу историю этих инструментов. Как часто бывает в случае с Unix, это история о том, как обмен информацией между программами, программистами и пользователями порождал цикл новаторских решений и усовершенствований.

В 1966 году на стажировке в MIT я столкнулся с программой Джерри Зальцера Runoff. Название восходит к выражению «I'll run off a copy for you» («Я отпечатаю тебе копию»). Это было простое средство форматирования текста. На ввод подавался обычный текст, перемежающийся начинающимися с точки строками, которые задавали способ форматирования. Например, этот раздел выглядел бы так:

```
.ll 60
.ce
Подготовка документов
.sp 2
.ti 5
В операционной системе Unix изначально были ...
.sp
.ti 5
В 1966 году на стажировке ...
```

Такая «разметка» давала программе понять, что делать с текстом. Устанавливалась длина строки (line length) в 60 символов, следующая строка центрировалась (center), шел пропуск (space) двух строк, временный отступ (temporarily indent) в пять пробелов, идущий после этого абзац выводится в виде строк размером не более 60 символов. Затем пропуск еще одной строки и отступ для следующего абзаца.

В Runoff был десяток или два подобных команд, позволяющих легко форматировать простые документы — страницы справочников, описания программ, письма друзьям. Словом, эти команды выполняли форматирование текста, легко достижимое в наши дни с помощью такого инструмента, как язык разметки Markdown.

РАННИЕ СРЕДСТВА ФОРМАТИРОВАНИЯ

Программа Runoff стала для меня откровением. Оказывается, компьютеры можно использовать не только для математических вычислений или компиляции. Они позволяли легко улучшать тексты при совсем небольших затратах. Современному читателю, скорее всего, сложно представить трудоемкость процесса подготовки документов до появления программ обработки текста. В нашем распоряжении были только механические пишущие машинки. Конечно, это лучше глиняных табличек или перьевых ручек, но для внесения в документ любых изменений требовалось заново печатать всю страницу. Поэтому большинство документов проверялось всего один или два раза, изменения в них вносились от руки, а потом все аккуратно перепечатывалось начисто.

Когда осенью 1968 года я начал писать диссертацию, то мечтал о программе Runoff, потому что альтернативой был самостоятельный набор на пишущей машинке (причем перепечатывалась вся страница в случае ошибок). Или нужно было кому-то заплатить, чтобы он напечатал все вместо меня. Печатаю я быстро, но неаккуратно, так что первый вариант отпадал. Нанять я тоже никого не мог, потому что был жадным и бедным.

Поэтому я написал простую версию программы Runoff, которую назвал Roff. Вот только в Принстоне не было ни интерактивной компьютерной системы, такой как CTSS, ни компьютерных терминалов. Доступны были перфокарты, поддерживающие только заглавные буквы. Программу Roff я написал на Фортране (вариант не лучший, так как Фортран предназначался для научных вычислений, а не для управления символами, но других попросту не было), добавив туда функцию преобразования только в нижний регистр, с автоматическим сохранением верхнего регистра для первой буквы в предложении. Полученный таким способом текст был распечатан на принтере IBM 1403, который умел печатать буквы в обоих регистрах. Вот так выглядели передовые технологии того времени! Моя диссертация

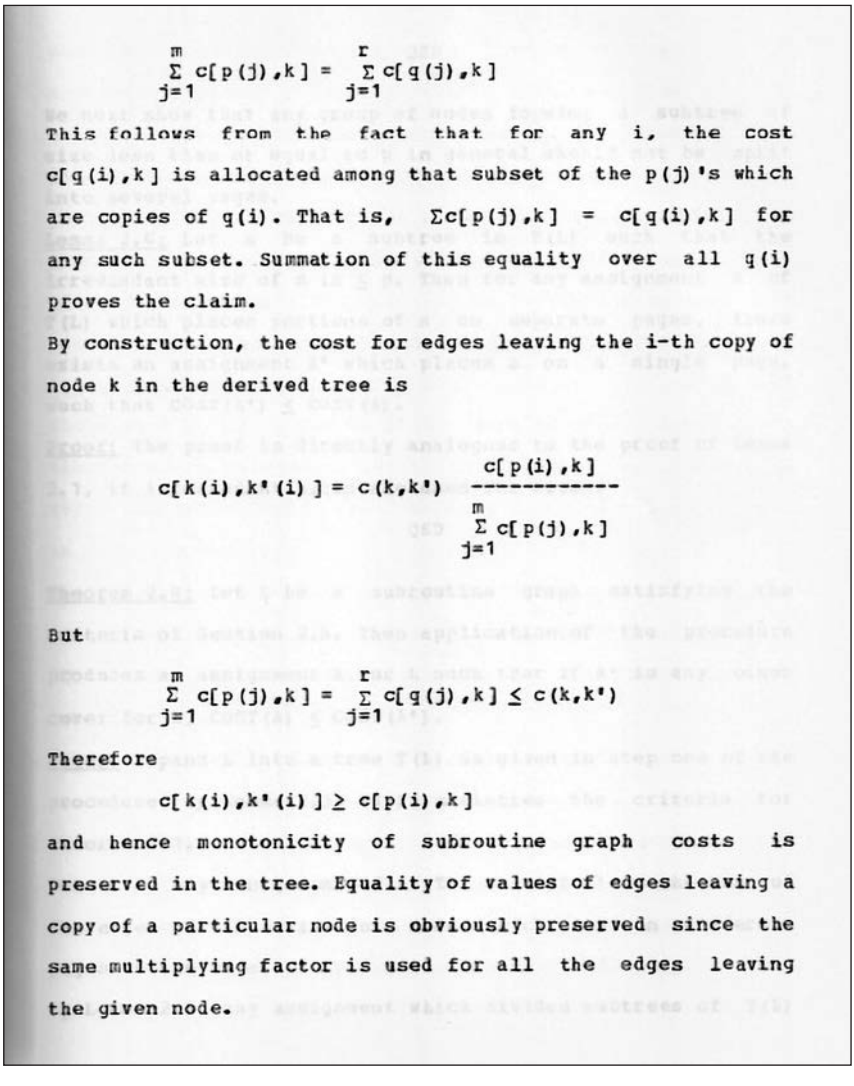
заняла три коробки, по две тысячи перфокарт в каждой. Длина коробки составляла примерно 35 см и весила 4,5 кг. Первые тысячу карт занимала программа, а остальные пять были отданы под собственно диссертацию в формате Roff.

Читателям, которые никогда не работали с перфокартами, подобное сложно представить. Одна карта содержала не более 80 символов, то есть туда можно было поместить или одну строчку кода на Фортране, или одну строку текста диссертации. Редактирование текста требовало пробивки новых перфокарт, которыми заменяли старый вариант. Для исправления орфографической ошибки обычно было достаточно заменить всего одну перфокарту, хотя, если новый текст оказывался длиннее, их могло потребоваться несколько.

На печатные страницы мне пришлось вручную добавлять некоторые специальные символы, например знаки суммирования (Σ), но в целом придуманный мной механизм сработал на удивление хорошо. Его оказалось достаточно для распечатки моей диссертации, которую можно считать первой диссертацией в Принстоне, напечатанной на компьютере. На рис. 5.6 вы видите одну из страниц оттуда. После этого несколько лет просуществовало студенческое агентство, которое за скромную плату осуществляло «roff» документов для студентов. Фактически Roff стала первой из написанных мной программ, которая принесла существенную пользу другим людям.

Попав в Bell Labs, я обнаружил, что там уже велась разработка пары подобных программ. В их числе была и программа Дуга Макилроя, базирующаяся на оригинальной программе Зальцера. Вскоре после этого Джо Оссанна написал более мощную версию, которую назвал Nroff (новый Roff). Она позволила патентному ведомству форматировать заявки на патенты. Как я уже упоминал, именно программа Nroff оказалась тем критически важным инструментом, который обеспечил приобретение первых компьютеров PDP-11 для работы над Unix.

Все это идеально соответствовало моим интересам. Я влился в маленькую группу людей, увлеченных идеей подготовки документов, и в сообщества активных пользователей таких программ, потратив большую часть следующего десятилетия на работу над инструментами форматирования текста.



$$\sum_{j=1}^{m} c[p(j),k] = \sum_{j=1}^{r} c[q(j),k]$$

This follows from the fact that for any i, the cost c[q(i),k] is allocated among that subset of the p(j)'s which are copies of q(i). That is, $\Sigma c[p(j),k] = c[q(i),k]$ for any such subset. Summation of this equality over all q(i) proves the claim.

By construction, the cost for edges leaving the i-th copy of node k in the derived tree is

$$c[k(i),k'(i)] = c(k,k') \frac{c[p(i),k]}{\sum_{j=1}^{m} c[p(j),k]}$$

But

$$\sum_{j=1}^{m} c[p(j),k] = \sum_{j=1}^{r} c[q(j),k] \leq c(k,k')$$

Therefore

$$c[k(i),k'(i)] \geq c[p(i),k]$$

and hence monotonicity of subroutine graph costs is preserved in the tree. Equality of values of edges leaving a copy of a particular node is obviously preserved since the same multiplying factor is used for all the edges leaving the given node.

Рис. 5.6. Страница моей диссертации, отформатированная в Roff

TROFF И НАБОРНЫЕ УСТРОЙСТВА

Программы Roff и Nroff обрабатывали только наборы символов фиксированной ширины (были моноширинными) и понимали только стандартные алфавитные символы, которые можно было ввести с телетайпа модели 37,

поэтому качество вывода было не очень высоким. Но в 1973 году Джо Оссанна договорился о покупке популярного в газетном производстве фотонабора CAT от фирмы Graphic Systems. Это должно было улучшить вид внутренних технических документов и помочь патентному ведомству с усовершенствованием патентных заявок.

Фотонабор CAT позволяет печатать латиницей, курсивом и полужирным шрифтом обычный пропорционально разнесенный текст, кроме того, там есть набор греческих букв и специальных математических символов. Текст печатался на длинных рулонах фотобумаги, которую требовалось проявлять с помощью вредных для здоровья реактивов. Эта технология предшествовала лазерным принтерам, до широкой доступности которых оставалось как минимум еще 10 лет. Кроме того, отпечатки были черно-белыми; недорогая цветная печать появилась только через несколько десятилетий.

Каждый шрифт представлял собой кусок 35-мм пленки с изображениями символов. Пленка устанавливалась на быстро вращающееся колесо. На колесе помещалось четыре шрифта по 102 символа каждый, что давало комбинацию из 408 символов. Когда над фотобумагой оказывался нужный символ, он освещался интенсивным лучом света. Были доступны 16 размеров.

Наборное устройство работало медленно. Для изменения размеров требовалось поворачивать турель объектива, неприятнее всего была необходимость работать с фотохимикатами, но качество вывода позволяло создавать документы профессионального уровня. Случалось, что представленные в журналы статьи из Bell Labs ставились под сомнение: из-за доскональной проработки создавалось впечатление, что это уже где-то публиковавшийся материал.

Для управления наборным устройством Джо значительно расширил программу Nroff, назвав новую версию Troff. «T» — это первая буква слова typesetter (наборное устройство), а название произносится как ти-рофф. При всей своей сложности и головоломности язык этой программы позволял выполнить любое форматирование, хотя редко у кого доставало квалификации и терпения справиться с этой задачей. По сути, это был аналог языка ассемблера, поэтому большинство предпочитало пользоваться пакетами макросов, инкапсулировавших обычные операции форматирования, такие как создание заголовков, рубрик, абзацев, нумерованных списков и прочего. Макросы были языком более высокого уровня над низкоуров-

невыми командами языка Troff. Мастером создания макросов был Майк Леск — автор широко используемого пакета `ms`. Из моих знакомых больше никто даже приблизительно не умел так хорошо пользоваться возможностями программирования на Troff.

После появления у нас наборного устройства, позволяющего выводить различные шрифты с пропорциональным интервалом и с достаточным количеством специальных символов, можно было задуматься и о наборе книг и внутренних технических документов. Первой набранной книгой стала наша с Биллом Плоджером *The Elements of Programming Style*, написанная в 1974 году. Она получилась несовершенной из-за отсутствия моноширинного шрифта для отображения кода программ, но в целом результат нас удовлетворил.

Одна из основных причин, по которой мы с Биллом решили самостоятельно заняться набором, заключалась в том, что в компьютерных программах часто появлялись ошибки на этапе подготовки к публикации. Имея возможность полностью контролировать весь процесс от набора до готовых к печати страниц, мы могли тестировать программы непосредственно из текста, которого не касались ни посторонний наборщик, ни корректор. В результате мы получили практически не содержащую ошибок книгу по программированию, что для того времени было крайне необычно. С тех пор я все время использовал этот вариант обработки; все мои книги создавались с помощью программы Troff или ее современной версии Groff. К счастью, наборные устройства с их дорогими и неприятными в обращении материалами давно ушли в прошлое. Сегодня достаточно корректно подготовить файл PDF и отправить его издателю или вывести на принтер.

EQN И ДРУГИЕ ПРЕПРОЦЕССОРЫ

Авторы из Bell Labs хотели создавать документы, в которых был бы не только обычный текст. Им требовалась возможность добавлять математические формулы, таблицы, рисунки, библиографические ссылки и т. д. В принципе, программа Troff позволяла это делать, но работать в ней было сложно и неудобно. Поэтому мы начали разрабатывать специальные языки, облегчающие работу с конкретными типами технических материалов. В сфере подготовки документов мы проходили тем же путем развития, которым уже прошли традиционные языки программирования.

Первым из этих языков специального назначения стал Eqn. Это еще и программа для набора математических выражений, которую мы с Лориндой Черри (рис. 5.7) написали в 1974 году. Ведь как научно-исследовательское учреждение Bell Labs выпускали множество технических документов, в основном для внутреннего пользования. И многие из них содержали математические формулы. В Bell Labs работали талантливые машинистки, умеющие воспроизводить формулы с помощью печатной машинки, но это был очень долгий процесс. Еще больше усилий требовало внесение правок.

Рис. 5.7. Лоринда Черри, примерно 1981 год (из архива Джерарда Хольцманна)

Лоринда прорабатывала идею инструмента для отображения математических формул, а я хотел, чтобы язык совпадал с произносимым вслух текстом. Подозреваю, что эта идея бродила у меня в подсознании, после того как в аспирантуре я несколько лет добровольно начитывал технические книги, создавая записи для слепых, и много часов потратил на произнесение математических формул.

Препроцессор Eqn прекрасно справлялся с простыми математическими выражениями. Например, сумму

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$$

он записывал так:

```
sum from i=0 to inf 1 over 2 sup i = 2
```

Машинистки, занимавшиеся набором математических формул, легко смогли освоить Eqn, и процесс набора заметно ускорился. Достаточно простой язык этой программы быстро изучили и физики, которые через некоторое время начали печатать документы и статьи, не прибегая к помощи машинисток. Eqn послужил одним из прототипов математического режима в разработанной Доном Кнутом системе компьютерной верстки TeX (1978), которая теперь является стандартом оформления математических книг. Eqn служит препроцессором для программы Troff. Вывод Eqn передается в Troff следующим образом:

```
eqn file | troff >typeset.output
```

Eqn распознает математические конструкции и переводит их в команды Troff, оставляя все остальное нетронутым. Такой подход привел к четкому разделению на два языка и две программы с разными задачами. К такой реализации нас с Лориндой подтолкнули физические ограничения PDP-11. Для добавления математической обработки в Troff попросту не хватало памяти. Эта программа уже имела максимально возможный размер. Да и Джо Оссанна вряд ли благословил бы нас на переработку Troff, даже если бы мы этого захотели.

Язык Eqn базируется на блочной модели: выражения строятся в виде набора подогнанных по размеру блоков. Например, дробь представляет собой числитель и знаменатель, центрированные друг относительно друга и разделенные линией. Выражения с индексами, такие как x_i, реализуются с помощью пары блоков, в которой второй блок имеет меньший размер и расположен ниже первого.

Для задания грамматических правил и добавления к ним семантики мы использовали недавно разработанный Стивом Джонсоном компилятор компиляторов Yacc. Eqn оказался первым языком, построенным с применением Yacc, который не был компилятором для традиционного языка. Лично я считаю, что без Yacc языка Eqn не случилось бы, потому что писать для него синтаксический анализатор вручную я не собирался. Его грамматика была слишком сложной, кроме того, она часто менялась в процессе наших с Лориндой экспериментов с синтаксисом. Наш опыт применения

Yacc показал, что наличие качественных инструментов позволяет делать вещи, реализация которых без них слишком сложна, а иногда и вообще немыслима.

Препроцессоры для подготовки к печати сложного материала различных типов оказались хорошей идеей. Вскоре после Eqn Майк Леск создал язык Tbl для форматирования сложных таблиц и язык Refer для форматирования библиографий. Эти вещи часто требовались в технической документации.

Многие из описанных в этом разделе программ относятся к препроцессорам, то есть к программам, преобразующим какой-то язык в форму, подходящую для последующей обработки. Компилятор Cfront точнее всего можно описать как объектно-ориентированный препроцессор для языка Си, который развился в язык C++. Это тот случай, когда препроцессор в конце концов ушел, поскольку его функциональность была поглощена последующей обработкой. Но куда чаще препроцессоры продолжали существование в виде отдельных программ, как в случае таких инструментов подготовки документов, как Eqn и Tbl. Другой пример — `bc`, препроцессор для `dc` (написанного Бобом Моррисом калькулятора для вычислений с произвольной точностью). Этот калькулятор принимал выражения в постфиксной нотации, сложной для новичков. Лоринда Черри написала оболочку `bc`, которая выполняла преобразование из традиционной инфиксной записи в постфиксную.

Препроцессоры дают массу преимуществ. Во-первых, новая реализация языка не ограничена существующим синтаксисом и может использовать совершенно другой стиль, как это происходило с различными препроцессорами Troff. Во-вторых, при ограниченной памяти в многоуровневые программы просто невозможно включить дополнительные функции, как это было в случае с Troff. Наконец, перед подачей на вход выходные данные препроцессора допускают другие виды обработки. В пакете подготовки документов я часто использовал сценарии `sed` для исправления наборов символов и пробелов. Вместе с Крисом Ван Виком мы написали программы для вертикального выравнивания страниц, меняющие вывод Troff до того, как он перейдет к драйверу устройства. Подобное было бы невозможно в рамках одной программы, но когда процесс представляет собой конвейер, в него легко добавлять новые этапы вперед, назад или в середину.

АППАРАТНО-НЕЗАВИСИМЫЙ TROFF

В 1977 году в возрасте 48 лет умер Джо Оссанна. В числе прочего после него остался исходный код Troff, который в то время насчитывал почти десять тысяч строк непостижимого кода на Си. Джо вручную преобразовал его из первоначальной версии на языке ассемблера. Комментариев там не было, зато присутствовали десятки глобальных переменных с двухбуквенными именами и различные неочевидные приемы, позволяющие втиснуть в программу как можно больше информации в условиях нехватки памяти. В защиту Джо могу сказать, что без всего этого он просто не смог бы упаковать всю функциональность Troff в 65 Кбайт. У PDP-11/45, на котором мы работали в то время, это был максимальный объем памяти, доступный для пользовательских программ.

Примерно через год я наконец набрался смелости заняться этим кодом и медленно и осторожно начал его обновление. Кроме руководства пользователя, не было ни комментариев, ни документации. Но самая большая проблема заключалась в зависимости кода от уже устаревшего к тому времени фотонабора CAT фирмы Graphic Systems.

В конце концов мне удалось найти все фрагменты, базирующиеся на особенностях CAT, и заменить их универсальным кодом, связанным с таблицами характеристик набора текста, таких как символьные наборы, размеры, шрифты и разрешения. Я изобрел язык, позволяющий программе Troff осуществлять вывод, настроенный на возможности конкретного наборного устройства. Простые драйверы преобразовали этот вывод во ввод для конкретных устройств. В результате появилась так называемая аппаратно независимая версия Ditroff, позволяющая применять для подготовки документов различные препроцессоры, особенно Pic, который умел использовать преимущества более высокого разрешения новых наборных устройств для рисования линий и фигур.

Одним из таких устройств был Linotron 202 от компании Mergenthaler. В теории это наборное устройство выглядело как то, что требовалось для замены CAT. Оно быстро работало, имело высокое разрешение, отображало символы, рисуя их на экране. Оно управлялось стандартным мини-компьютером модели Naked Mini от фирмы Computer Automation, простой программой, похожей на написанные мной программы для других наборных устройств. Основным недостатком была стоимость — 50 тысяч долларов, но мы показали хорошие результаты с предыдущим наборным устройством, так что руководство без проблем одобрило эту покупку.

Когда мы получили Linotron 202, оказалось, что хуже его ненадежной аппаратной части было только его программное обеспечение. В результате к нам несколько месяцев почти ежедневно приходили представители ремонтной службы из компании Mergenthaler, а Кен Томпсон и Джо Кондон (рис. 5.8) занимались обратной разработкой.

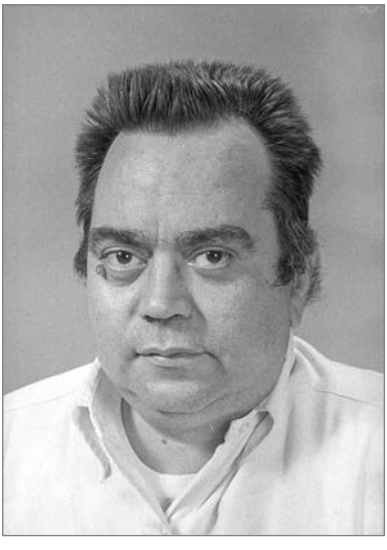


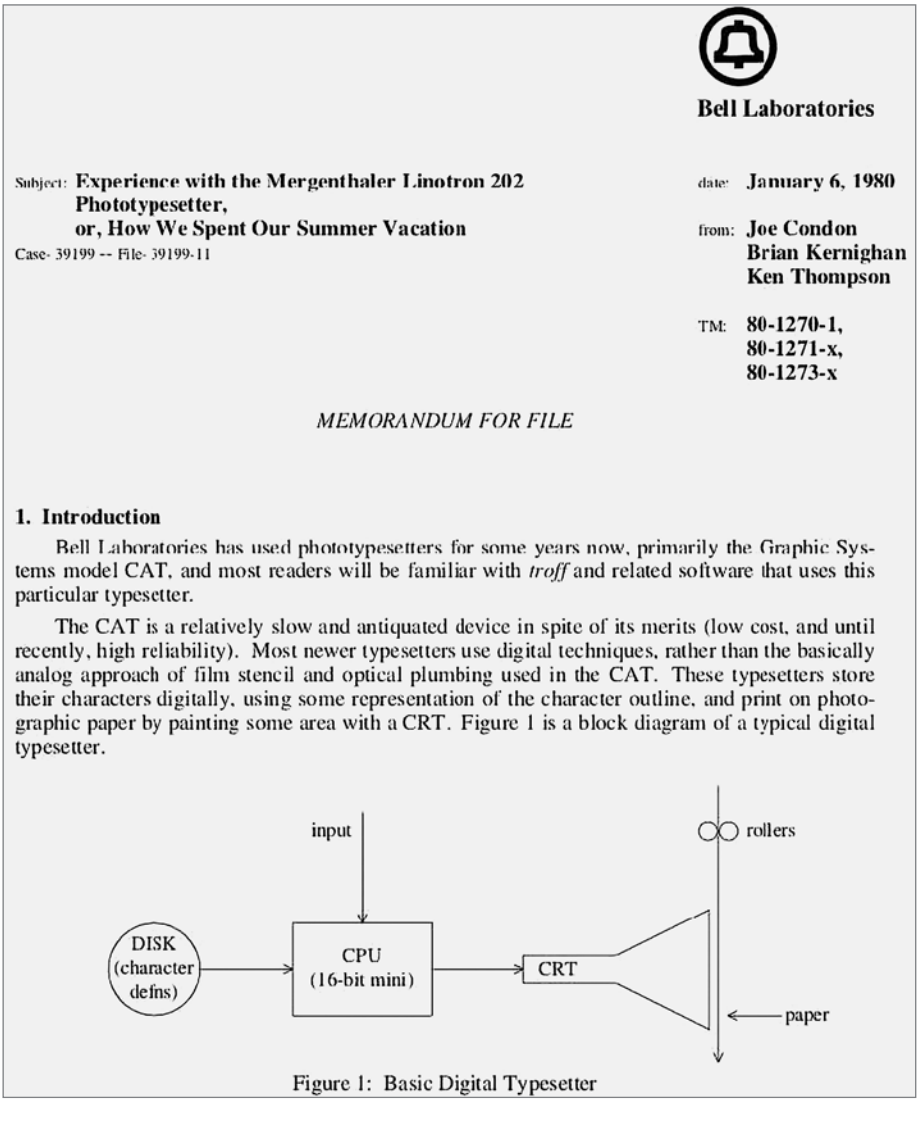
Рис. 5.8. Джо Кондон, примерно 1981 год (из архива Джерарда Хольцманна)

По образованию Джо был физиком, но по мере изменения его интересов перешел к разработке электронных схем и написал множество инструментов для их проектирования, которые Центр использовал для экспериментов с аппаратным обеспечением. Вместе с Кеном Джо разрабатывал шахматную машину Belle. Именно его опыт помог разобраться с Linotron 202.

Кен начал с написания дизассемблера двоичных программ, которые запускались на этой машине. Хочу отметить, что он справился с этим за пару часов. Я пошел домой поужинать, а когда вернулся, все было готово.

Дизассемблирование программ фирмы Mergenthaler дало Кену и Джо представление о том, как функционирует наборное устройство. После нескольких недель напряженного инженерного анализа они разобрались со способом кодирования символов в фирме Mergenthaler. И написали код, позволяющий создавать наши собственные символы: символ системы

Белла, который вы видите в верхней части рис. 5.9, шрифт chess для печати шахматных диаграмм и лицо Питера, которое мы использовали разными способами (один из примеров показан на рис. 5.10).



Bell Laboratories

Subject: **Experience with the Mergenthaler Linotron 202 Phototypesetter, or, How We Spent Our Summer Vacation**

Case- 39199 -- File- 39199-11

date: **January 6, 1980**

from: **Joe Condon**
Brian Kernighan
Ken Thompson

TM: **80-1270-1, 80-1271-x, 80-1273-x**

MEMORANDUM FOR FILE

1. Introduction

Bell Laboratories has used phototypesetters for some years now, primarily the Graphic Systems model CAT, and most readers will be familiar with *troff* and related software that uses this particular typesetter.

The CAT is a relatively slow and antiquated device in spite of its merits (low cost, and until recently, high reliability). Most newer typesetters use digital techniques, rather than the basically analog approach of film stencil and optical plumbing used in the CAT. These typesetters store their characters digitally, using some representation of the character outline, and print on photographic paper by painting some area with a CRT. Figure 1 is a block diagram of a typical digital typesetter.

Figure 1: Basic Digital Typesetter

Рис. 5.9. Неопубликованная служебная записка о борьбе с наборным устройством Linotron 202

Рис. 5.10. Таблица проверки зрения с лицом Питера (из архива Джерарда Хольцманна)

Кен написал для контроллера Mergenthaler интерпретатор языка B, а мы — программы на B для управления этим контроллером. Подробно эта история изложена в технической записке, которую руководство Bell Labs скрыло, вероятно, для охраны какой-то интеллектуальной собственности Mergenthaler. Она была опубликована только в 2013 году (см. материалы по адресу https://www.cs.princeton.edu/~bwk/202/). На фрагменте первой страницы, который показан на рис. 5.9, обратите внимание на неполные внутренние номера документа, такие как 80-1271-x. Это говорит о том, что документу так и не был присвоен официальный номер.

Когда в конечном итоге наборное устройство заработало, мы смогли благодаря высокому разрешению создавать интересные графические эффекты, включая полутоновые изображения и чертежи, такие как схема цифровой наборной машины на рис. 5.9. Для чертежей я создал язык Pic, позволяющий представлять в виде текста рисунки, такие как организационные диаграммы или диаграммы сетевых пакетов. Естественно, обработку грамматики выполнял препроцессор Yacc, а за лексическую часть отвечал Lex. Простой пример возможностей языка Pic показан на рис. 5.11.

```
arrow "stdin" above
box "command"
arrow "stdout" above
arrow down from last box.s "     stderr" ljust
```

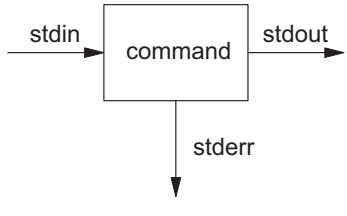


Рис. 5.11. Язык для набора графики Pic (ввод и вывод)

ПУБЛИКАЦИЯ КНИГИ

Инструменты подготовки документов хорошо работали благодаря тому, что мы активно ими пользовались. Если программа функционировала некорректно или в ней возникала ошибка, можно было подойти к автору кода и попросить внести исправления или добавить нужную функциональность.

Причем это касалось не только программного обеспечения для подготовки документов. Все мы пользовались собственноручно написанными программами, что давало реальный стимул к их улучшению.

Члены Исследовательского центра компьютерных наук за 1970-е и 1980-е написали множество книг, оказавших заметное влияние на вычислительную технику и компьютерные науки. Этих книг оказалось намного больше, чем можно было ожидать от промышленной исследовательской лаборатории.

Аль Ахо написал несколько учебников, которые приобрели широкую известность. В их числе знаменитая «Книга Дракона» 1977 года, написанная в соавторстве с Джеффом Уллманом («Компиляторы: принципы, технологии и инструменты»). Обложка ее первого издания показана на рис. 5.12. Затем вместе с Джеффом и Джоном Хопкрофтом была написана книга «Структуры данных и алгоритмы». В начале 1980-х Бьёрн Страуструп (рис. 5.13) создал язык C++, а несколько лет спустя написал про него

Рис. 5.12. Первое издание «Книги дракона» Ахо и Уллмана, 1977 год

Рис. 5.13. Бьёрн Страуструп, примерно 1984 год
(из архива Бьёрна Страуструпа)

несколько книг. Книга Джона Бентли «Жемчужины программирования» выросла из его статей для журнала *Communications of the ACM*. Майк Гэри и Дэвид Джонсон из математического центра использовали программы Troff и Eqn для подготовки своей ключевой книги *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Еще мы выпустили в форме книг руководства по Unix, Plan 9 и т. п. Они стали базовыми учебниками и справочниками для поколений программистов и студентов, изучающих информатику.

Каким же образом сравнительно небольшой группе исследователей удалось выпустить так много значимых книг?

Во-первых, люди относились к делу серьезно, они изо всех сил старались написать качественный текст и были критиками работ своих коллег. Первое место в этой группе занимал Дуг Макилрой. Никто другой не обладал такой способностью обнаруживать в любом тексте независимо от его темы ошибки (от крошечных до критических) и обращать внимание на не до конца понятные фрагменты. Я всегда просил Дуга прокомментировать написанное мной, и он никогда не отказывал. Было унизительно наблюдать,

как он рвет мои тексты, но именно это научило меня писать намного лучше. У других была такая же ситуация.

Конечно, Дуг был не единственным критиком. Любой мог уделить этому время. Подробно комментировать написанное коллегами было частью нашей культуры. Такая необычная практика являлась одним из того, что делало Bell Labs отличным местом для работы.

Во-вторых, написание книг поддерживало местное руководство. Любые публикации, в том числе книги, поддерживали репутацию Bell Labs в научных и академических кругах. Руководство давало возможность в течение шести месяцев целыми днями работать над книгой. Такого концентрированного усилия хватало для завершения работы, которая в противном случае могла растянуться на несколько лет. И это еще не все. Хотя авторские права на книги оставались у Bell Labs, авторы получали гонорары.

Я сомневаюсь, чтобы кто-то из нас хоть раз писал книгу с целью заработка. Все сотрудники Bell Labs понимали, что особой выгоды написание технических книг не принесет. Но если книга начинала пользоваться успехом, деньги от ее продажи шли автору. Такое грамотное руководство и политика компании побуждали людей писать, что в итоге окупалось для всех участвующих сторон. Публикации авторов из Bell Labs помогали привлекать новых людей.

Деятельность Bell Labs не была секретом. Студенты знали, что именно тут создавалось программное обеспечение, которым они пользуются, и были написаны учебники, по которым они учатся. Потенциальные новые сотрудники видели, что результаты хорошо проделанной работы публикуются. Можно было не беспокоиться, что все исчезнет в «промышленной» исследовательской лаборатории. Это ставило Bell Labs на одну доску с университетами, но с тем преимуществом, что исследованиями можно было заниматься на полную ставку, не отвлекаясь на преподавание, административные задачи и изыскание средств. Именно комбинация отличного программного обеспечения и значимых книг по большей части обеспечила успех Bell Labs.

В-третьих, присутствовали и такие технические факторы, как симбиоз между языком Си и средой программирования Unix, подготовка документов как область исследований и написание технических компьютерных текстов в качестве основной деятельности. Все началось с программ форматирования текста, таких как Roff Дуга Макилроя, Nroff Джо Оссанны

и Troff, а также с препроцессоров, таких как Eqn, Tbl и т. д. Эти инструменты облегчали создание документов со все более сложными типографскими материалами — математическими формулами, таблицами, рисунками, диаграммами и графиками. Все это положительно сказывалось на качестве текстов, ведь все программы подготовки документов позволяли легко вносить многочисленные правки и всегда иметь чистую рабочую копию, что сильно отличалось от изначальной ситуации, когда материалы отдавались машинистке и приходилось днями ждать их возвращения.

Возможно, это звучит тривиально, но я уверен, что качество материалов обеспечивала именно легкая возможность редактирования. Потому что она в значительной степени исключала накладные расходы на посредников — машинисток, редакторов и полиграфистов. Для технических документов и для Руководства программиста Unix была важна точность, но в случае с книгами на первое место вышел контроль всего процесса. Для книг по программированию жизненно важной была возможность набирать программы непосредственно из исходного кода. Только это давало уверенность в том, что код будет напечатан правильно, что его не смогут случайно изменить посторонние люди.

Все инструменты, конечно, были написаны на Си, так как это был выразительный и эффективный язык. Сегодня уже мало кто знает и помнит, что в те времена, когда емкость запоминающих устройств компьютеров выражалась в килобайтах, а не в гигабайтах, эффективность как во времени, так и в пространстве имела решающее значение. Подсчитывался каждый байт, имело значение, на каком уровне выполнялась каждая инструкция, поэтому язык, позволяющий учитывать оба этих аспекта, был практической необходимостью.

Все это прошло полный цикл развития. Для подготовки этой книги применялись потомки упоминавшихся выше программ подготовки документов — Groff, Geqn и т. п., которые появились благодаря отличным свежим реализациям и улучшениям, написанным Джеймсом Кларком.

5.4. SED И AWK

Одним из основных упрощений файловой системы Unix стала единообразная обработка файлов как последовательностей неинтерпретиру-

емых байтов. Исчезли записи, обязательные или запрещенные символы и навязываемая файловой системой внутренняя структура. Учитывались только байты.

Упрощение коснулось и способа обработки текстовых данных в большинстве программ Unix. Текстовые файлы воспринимались как последовательности байтов, которые оказались символами в кодах ASCII (American Standard Code for Information Interchange — Американский стандартный код для обмена информацией). Такое единообразное представление текста естественным образом вписалось в концепцию конвейеров. Инструментарий Unix включал множество программ, которые считывали ввод текста, что-то с ним делали и записывали вывод. Я уже упоминал такие примеры, как подсчет слов, сравнение, сортировка, транслитерация, поиск дубликатов и, конечно, поиск с помощью утилиты grep.

SED

Успех утилиты grep вдохновил Ли Макмэхона написать аналогичную программу `gres`, которая в процессе просмотра текста делала простые замены. В текстовом редакторе `ed` «S» была командой замены. Вскоре Ли написал более универсальную версию — потоковый редактор Sed, который применял предопределенные преобразования к потоку текстовых данных. Утилиты grep и `gres` представляют собой частные случаи редактора Sed. Набор команд в Sed сделан по образцу редактора `ed`. Сейчас Sed часто применяется в сценариях оболочки для преобразования потока данных определенным образом: замены или добавления символов, удаления ненужных пробелов или других нежелательных символов.

Ли обладал нестандартной квалификацией и опытом работы. Прежде чем заняться компьютерными науками, он получил в Гарварде степень по психологии и даже обучался в иезуитской семинарии, чтобы стать священником. Одним из первых в группе Unix он задумался об обработке больших объемов текста, причем случилось это в те времена, когда оперативная память была слишком ограниченна, чтобы хранить такие объемы. Впрочем, тут можно вспомнить, что понятие «большой» относительно. Особый интерес у Ли в начале 1970-х годов вызывал сборник из 85 статей в поддержку ратификации Конституции США (так называемые «Записки федералиста»), которые в совокупности занимали чуть больше мегабайта.

AWK

Меня интересовали инструменты, одинаково хорошо умеющие обрабатывать как числа, так и текст. Как grep, так и Sed не умеют обрабатывать числовые данные и выполнять арифметические действия, кроме того, grep не работает с наборами текстовых строк; для таких вещей все еще требовалась программа на Си. Поэтому я искал какой-то обобщенный вариант. В то же время Аль Ахо (рис. 5.14), экспериментируя с регулярными выражениями, которые не поддерживались в grep, написал утилиту egrep («расширенный grep»). Кроме того, в подразделение 1127 перешел Питер Вайнбергер, занявший кабинет между Алем и мной. Его интересовали базы данных.

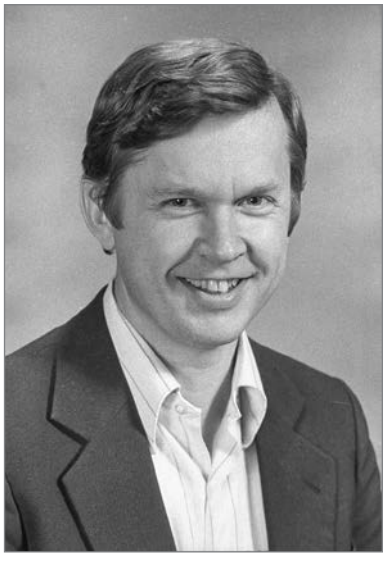


Рис. 5.14. Аль Ахо, примерно 1981 год (из архива Джерарда Хольцманна)

Осенью 1977 года мы втроем искали способ объединить все эти вещи, черпая вдохновение из языка RPG, мощного, но непостижимого генератора программ отчетов от корпорации IBM, а также из изящной идеи Марка Рочкинда, о которой я расскажу в следующей главе. В конечном итоге мы разработали язык AWK (далее Awk). Как говорилось в оригинальном описании, именование языка в честь его авторов показывает некоторую нехватку воображения. Я уже не помню, рассматривали ли мы аналогию со словом «awkward» (неуклюжий, громоздкий, неудобоваримый), но в любом

случае название прижилось. Первую версию Питер написал очень быстро, всего за несколько дней, используя Yacc, Lex и код регулярных выражений из утилиты Аля egrep.

Программа на Awk представляет собой последовательность шаблонов и действий. Каждая строка ввода сравнивается с каждым шаблоном, и при обнаружении совпадения выполняется соответствующее действие. В качестве шаблонов могут выступать как регулярные выражения, так и числовые или строковые относительные выражения. Действия написаны на диалекте Си. Если шаблон не указан, действие выполняется для любой записи. Если не указано действие, происходит вывод записи.

Вот пример команды, выводящей все входные строки, длина которых превышает 80 символов; это шаблон, для которого не указано действие.

```
awk 'length > 80'
```

Awk поддерживает числовые и строковые переменные, а также ассоциативные массивы, чьи индексы могут быть числами или произвольными строками символов. Переменные инициализируются нулем и пустой строкой, поэтому обычно в присваивании начальных значений необходимости нет.

Awk автоматически читает каждую строку каждого входного файла и разбивает ее на поля, поэтому код для явного чтения ввода или анализа отдельных строк требуется редко. Есть также встроенные переменные, которые содержат номер текущей входной строки и количество полей в ней, поэтому эти значения вычислять не нужно. Такие значения по умолчанию исключают шаблонный код и обеспечивают многим программам на Awk длину в одну или две строки.

Например, вот программа, которая ставит в начало каждой строки ее порядковый номер:

```
awk '{print NR, $0}'
```

Здесь `NR` — это номер текущей входной строки, а `$0` — сама входная строка.

А вот канонический пример подсчета количества вхождений каждого слова и вывода слов и их количества:

```
{ for (i = 1; i <= NF; i++) wd[$i]++ }
END { for (w in wd) print w, wd[w] }
```

В первой строке действие без шаблона; соответственно, оно выполняется для всех входных строк. Встроенная переменная `NF` — автоматически вычисляемое количество полей в текущей строке ввода. Переменная `$i` предназначена для поля с номером `i`, ее значение тоже вычисляется автоматически. Оператор `wd[$i]++` использует это значение в качестве индекса массива `wd`, увеличивая этот элемент массива. Совпадение со специальным шаблоном `END` ищется после чтения последней строки ввода. Обратите внимание на два типа цикла `for`. Первый заимствован непосредственно из Си. Второй циклически перебирает элементы массива. В рассматриваемом примере он выводит строку для каждого отдельного слова исходного ввода вместе со счетчиком того, сколько раз в тексте появляется это слово.

Хотя множество задач в какой-то момент начали решаться средствами языка Perl, а позже и языка Python, Awk продолжает широко применяться и сегодня. Кроме основной версии существуют как минимум четыре или пять независимых реализаций, в том числе Gawk Арнольда Роббинса и Mawk Майкла Бреннана. Конечно, в Awk встречаются сомнительные проектные решения и неясные места, но, на мой взгляд, он самый выгодный среди всех языков программирования: большую часть можно выучить за 5–10 минут, а типичные программы состоят всего из нескольких строк. Масштабируется он плохо, что не мешает людям писать на Awk программы длиной в тысячи строк.

Потоковый редактор Sed также популярен как частый компонент конвейерных сценариев командной оболочки. У меня даже есть наклейка на бампер с надписью:

Sed and Awk: вместе мы можем изменить все.

Стоит отметить, что языки Sed, Awk, Make, Yacc и Lex воплощают разновидности парадигмы шаблон — действие. Программы на этих языках состоят из последовательности шаблонов и действий; основная операция заключается в проверке входных данных на соответствие каждому шаблону, и в случае совпадения выполняется указанное действие. Шаблоны и действия можно опускать, тогда имеет место поведение по умолчанию.

Например, grep, Sed и Awk могут использоваться для сопоставления с одним регулярным выражением. Следующие три команды эквивалент-

ны при условии, что для каждой из них допустимо заданное регулярное выражение:

```
grep re
sed -n /re/p
awk /re/
```

Парадигма шаблон — действие — это естественный способ представления вычислений, сводящихся в основном к последовательности проверок и действий.

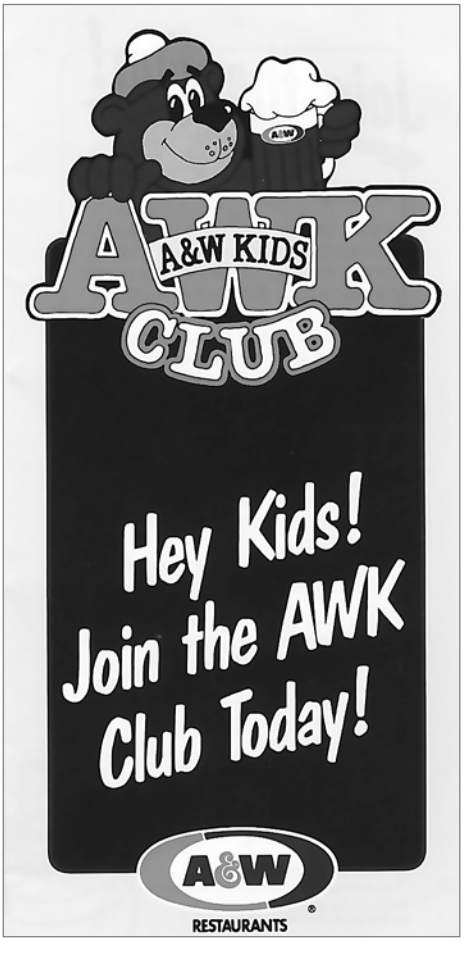


Рис. 5.15. Awk в массовой культуре

5.5. ДРУГИЕ ЯЗЫКИ

Среда программирования Unix, ее инструменты для разработки языков, множество потенциальных областей применения и, конечно же, наши эксперты, разбирающиеся во всем, от компиляторов до теории и алгоритмов языка, обеспечили благодатную почву для появления различных языков. Я не буду подробно рассматривать эту тему, но упомянуть некоторые из языков стоит.

Ничего страшного, если вы не сможете понять все тонкости примеров, которые будут приведены ниже. Основной смысл заключается в том, что широкие интересы, компетентность в сфере языков и Yacc, и Lex позволяли сотрудникам Центра относительно легко создавать языки для новых областей применения. Без комбинации перечисленных факторов это потребовало бы намного больших усилий, и думаю, что многих интересных языков сейчас не существовало бы.

Наиболее значимым примером может служить C++, появившийся в 1979 году, когда к подразделению 1127 присоединился только что получивший научную степень в Кембридже Бьёрн Страуструп. Бьёрна интересовали симуляции и операционные системы, а существующие языки не удовлетворяли его потребностей. Поэтому он взял несколько хороших идей из максимально подходящего для его целей языка Simula и скомбинировал их с Си. Результат, объединивший идеи объектно-ориентированного программирования с эффективностью и выразительностью Си и названный «Си с классами», датируется 1980 годом.

Сочетание оказалось удачным, и язык стал пользоваться успехом. В 1983 году появилось название C++, намек Рика Маскитти на оператор унарного постфиксного инкремента. Сегодня C++ — один из наиболее широко используемых языков программирования, лежащий в основе пакета Microsoft Office, больших фрагментов инфраструктуры Google, вашего любимого браузера (каким бы он ни был), многих видеоигр и прочего скрытого от глаз пользователей программного обеспечения.

Бьёрн работал в моем отделе около 15 лет и, как я уже упоминал, часто заходил обсудить проектные решения, так что процесс эволюции C++ я наблюдал с самого начала. По крайней мере, я хорошо разбирался в этом языке на ранней стадии, но сейчас, когда он сильно разросся, понимаю его с трудом.

C++ часто критикуют за размер, а иногда и за унаследованный от Си синтаксис. Но из наших многолетних разговоров я знаю, что в языке нет ни одной особенности, для которой не существовало бы веской причины. Также разумным инженерным и маркетинговым решением было сделать C ++ расширенной версией языка Си, хотя для этого требовалось учитывать множество синтаксических и семантических узких мест последнего. Но если бы не стремление Бьёрна к совместимости с Си, шансы C++ на успех были бы гораздо меньше. Создать новый язык очень трудно, принципиальную важность имеет совместимость как на уровне исходного кода (чтобы пользователи видели уже знакомый синтаксис), так и на уровне объектов (чтобы можно было пользоваться существующими библиотеками Си). Именно эти факторы сделали новый язык таким же эффективным, как Си.

Впрочем, в подразделении 1127 родились и другие, не менее значимые языки.

Стю Фельдман и Питер Вайнбергер написали `f77` — первый компилятор языка Fortran 77. Версия Fortran 77 несколько превосходила Fortran 66, для которого я написал препроцессор Ratfor, хотя и в ней не было достаточного количества операторов для управления ходом выполнения программы. В любом случае создать `f77` было непросто, но оно того стоило, поскольку в подразделении 1127 им активно пользовались специалисты по численному анализу, работающие как на PDP-11, так и на VAX.

Поэтому Стю и Дейв Гэй написали программу `f2c`, преобразующую код на Фортране в код на Си. Это дало возможность пользоваться Фортраном в системах, где компилятор этого языка отсутствовал или был дорогим коммерческим продуктом.

Пришедший в подразделение 1127 из Делфтского технического университета Джерард Хольцманн (рис. 5.16) всегда интересовался фотографией. В начале 1980-х у него появилась идея языка программирования для алгоритмического преобразования файлов цифровых изображений. Он назвал этот язык Pico:

Первоначально название намекало на его размер, позже его было проще воспринять как аббревиатуру словосочетания picture composition (композиция изображения).

Рис. 5.16. Джерард Хольцманн, примерно 1981 год
(из архива Джерарда Хольцманна)

Язык Pico тоже работает по принципу шаблон — действие. Он определяет новые изображения, однократно вычисляя для каждого пикселя исходного изображения заданное пользователем выражение. Выражения могут ссылаться на значения, координаты, различные функции и части других изображений. Таким способом можно выполнять интересные преобразования (один из примеров показан на рис. 5.17), многие из которых описаны в книге *Beyond Photography*, опубликованной Джерардом в 1988 году. Неудивительно, что Pico был реализован на Си с генератором синтаксических анализаторов Yacc.

Рис. 5.17. Преобразованное с помощью Pico изображение Джерарда Хольцманна

Еще Джерард создал специализированный инструмент Spin для анализа и проверки правильности программных систем, в которых происходят процессы обмена данными. Эта утилита позволяет проверять логическую корректность конкретной системы и отсутствие в ней дефектов, таких как ситуации взаимной блокировки. («После вас». «Нет, только после вас».) Это отличный пример того, к какому результату привели проводившиеся в подразделении 1127 исследования способов представления взаимодействия отдельных процессов. Наряду с некоторыми первоклассными разработками программного обеспечения появилась легкая в реализации и использовании и достаточно быстродействующая система. Модели программ в Spin пишутся на языке специального назначения Promela (protocol metalanguage), также построенном с помощью Yacc.

Пакет Spin установлен на многих тысячах компьютеров, для его пользователей ежегодно проводятся конференции. Он применяется для проверки множества систем, от разработки аппаратного обеспечения до протоколов железнодорожной сигнализации.

Вместе с Бобом Фоурером и Дейвом Гэем мы разработали и внедрили AMPL — язык для описания задач оптимизации, таких как линейное программирование. Боб был профессором в области теории управления и исследования операций в Северо-Западном университете. Он давно искал способы помощи с созданием математических моделей оптимизации. Наша работа над AMPL началась, когда в 1984 году во время творческого отпуска он появился в Bell Labs.

Язык AMPL позволяет легко определять модели, описывающие конкретные задачи оптимизации. Например, поиск самого лучшего способа доставки товаров с заводов в магазины, определение затрат на доставку, прогноз продаж в каждом магазине, производственные мощности каждого завода и т. п. Задачи оптимизации записываются в алгебраической нотации. Они формулируются для набора ограничений и для целевой функции, для которой нужно найти экстремум, то есть максимум или минимум.

Задачи оптимизации играют важную роль во многих отраслях: планирование полетов экипажей различных авиакомпаний, производство, доставка и распределение, товарный учет, рекламные кампании и множество других сфер. Первоначальную реализацию AMPL я сделал на C++ вместе с генератором Yacc, который отвечал за синтаксис, и анализатором Lex,

ответственным за лексическую часть. Это была моя первая серьезная программа на C++, но вскоре я передал ее код Дэйву Гэю.

Наверное, AMPL — это единственный широко используемый язык, созданный в подразделении 1127, который стал собственностью компании. Сам язык невозможно защитить авторским правом, но мне пока не известны его реализации с открытым исходным кодом. Через несколько лет после его создания AT&T начала продавать компаниям лицензии на AMPL. После нашего с Дейвом ухода из Bell Labs мы с ним и еще одним коллегой открыли небольшую компанию AMPL Optimization для продолжения разработки и продвижения языка AMPL. В конце концов мы выкупили у Bell Labs права на него, чтобы обрести самостоятельность. Наша компания невелика, но играет значительную роль в своей рыночной нише.

В начале 1980-х Роб Пайк (рис. 5.18) и Лука Карделли экспериментировали с языками для многопоточного программирования, главным образом для взаимодействия с устройствами ввода, такими как мышь и клавиатура. Это породило такие языки, как Squeak и Newsqueak. Идеи Newsqueak в конечном итоге нашли свое отражение сначала в языках Limbo и Alef, которые использовались в операционной системе Plan 9, а десятилетие спустя — в языке программирования Go, разработанном внутри компании Google в 2008 году Робом Пайком, Кеном Томпсоном и Робертом Гризмером.

Рис. 5.18. Роб Пайк, примерно 1981 год (из архива Джерарда Хольцманна)

5.6. ДРУГИЕ ДОСТИЖЕНИЯ

До сих пор основное внимание уделялось системному программному обеспечению, особенно языкам, поскольку с этой темой я знаком лучше всего, но нельзя не упомянуть и о достижениях в области научных вычислений, коммуникаций, безопасности и аппаратного обеспечения. Ведь все они оказали свое влияние на развитие операционной системы, и с каждым из них связаны важные программные компоненты. Впрочем, не все они попадают именно в седьмую редакцию.

НАУЧНЫЕ ВЫЧИСЛЕНИЯ

Как и следовало ожидать, в Bell Labs рано начали применять компьютеры для моделирования и симуляции физических систем и обработки данных, что естественным образом вытекало из математических исследований. Собственно, прогноз Дика Хэмминга о вытеснении лабораторий компьютерными технологиями полностью подтвердился. Основное внимание уделялось численным методам линейной алгебры, дифференциальным и интегральным уравнениям, аппроксимации функций и математическим библиотекам, обеспечивающим широкую доступность самых известных методов.

Пионером в таких численных методах стала Филлис Фокс, которая внесла основной вклад в библиотеку PORT для программистов на Фортране. Эта библиотека определяла машинно-зависимые константы для диапазонов чисел, которые различались для разных компьютеров. Библиотеки гарантировали переносимость кода Фортран.

Библиотека PORT огромна. Это 130 тысяч строк на Фортране в 1500 программах и огромное количество документации. Барбара Райдер и Стю Фельдман разработали компилятор Фортрана PFORT, который проверял код на Фортране на наличие несовместимых расширений. Норм Шрайер написал программу проверки арифметических операций, так как результат на разных компьютерах зависел от способа выполнения операций с плавающей запятой. Это было особенно важно, поскольку предшествовало разработке стандартов поведения чисел с плавающей запятой.

Эрик Гросс и Билл Кофран придумали алгоритмы моделирования и симуляции полупроводников, анализ электронных схем и визуализацию,

актуальную для проектирования и изготовления полупроводников. Большая часть разработанных в Bell Labs программ для научных вычислений распространялась по миру через репозиторий Netlib, который до сих пор широко используется в научном сообществе. Среди специалистов по численному анализу, которые внесли значительный вклад в Netlib, следует упомянуть Дейва Гэя, Линду Кауфман и Маргарет Райт.

СПРАВОЧНИК НОМЕРОВ БЕСПЛАТНОГО ВЫЗОВА ОТ AT&T

Опыт Эрика Гросса в распространении программного обеспечения пригодился при реализации интересного дополнительного проекта. В 1994 году мы с Эриком и Лориндой Черри разместили в Интернете телефонный справочник номеров с кодом 800 от AT&T. Таким способом AT&T пыталась получить опыт в предоставлении реального интернет-сервиса (как и опыт работы с интернетом в принципе), а возможно, обеспечить дополнительные звонки на номера бесплатного вызова и в конечном итоге получить доход за счет расширенных услуг, таких как показ рекламы.

Кроме того, мы надеялись, что предоставление реально ценного сервиса, а не просто тизерной рекламы, которой пестрел интернет того времени, положительно скажется на образе компании.

В августе 1994 года мы получили снимок базы данных из 157 тысяч записей размером около 22 МБ и через несколько часов запустили его на локальном компьютере в виде веб-сайта с возможностью поиска и просмотра. Куда больше времени ушло на то, чтобы убедить руководство AT&T предоставить к нему общий доступ. Но после долгих дискуссий нежелание и инертность руководства были побеждены. Тем более прошел слух, что какую-то интернет-услугу собирается предложить конкурент AT&T компания MCI. В результате 19 октября 1994 года каталог стал общедоступным. Это был первый веб-сервис от AT&T.

Конечно, эта тактическая задержка была огорчительной, но в итоге история преподнесла нам урок. База данных оказалась наводнена ошибками, в ней было множество списков, которые никто не просматривал критическим взглядом; например, название города Цинциннати писалось девятью способами. Это могло служить хорошим примером регулярного выражения.

Несмотря на недостатки, телефонный справочник принес нам некоторую выгоду: одно время он шел первым в списке «популярных страниц» от

Yahoo. Сам проект Yahoo стартовал в начале 1994 года, и его индексирование было полностью ручным. Так AT&T одержала небольшую победу над MCI, рискнув первыми предоставить полезный сервис. С телефонного справочника началось представительство AT&T в интернете, что привело к множеству внутренних обсуждений вариантов дальнейших услуг. Как ничто другое, этот опыт заставил почувствовать удивительную скорость роста и изменения интернета. В своем неофициальном отчете я писал: «Если мы хотим быть представлены в этой области, следует научиться действовать быстро».

ПРОГРАММА UUCP

В середине 1970-х Майк Леск написал программу UUCP, которая осуществляла отправку файлов из одной системы Unix в другую, как правило, через обычную телефонную линию. Это был медленный, а иногда и дорогой способ связи, зато широко распространенный. Большинство Unix-систем того времени имели коммутируемый доступ, хотя пользовались им немногие, так как приходилось платить за телефонные звонки.

В основном программа UUCP применялась для распространения программного обеспечения, но со временем она легла в основу удаленного выполнения команд, передачи почты и групп новостей. Причем все это произошло задолго до широкой доступности интернета. На базе UUCP работала одна из первых в мире систем распространения информации — компьютерная сеть Usenet. Первый дистрибутив UUCP включили в седьмую редакцию. Через несколько лет его усовершенствовали, перенесли на другие операционные системы и превратили в ПО с открытым исходным кодом. Применение UUCP прекратилось после того, как интернет стал стандартной сетью связи с собственными протоколами.

БЕЗОПАСНОСТЬ

Безопасность волновала членов сообщества Unix с первых дней. Частично этот интерес был унаследован от Multics, а частично возник благодаря опыту в криптографии.

Одна из задач состояла в предоставлении пользователям файловой системы возможности контролировать доступ к своим файлам. В файловой системе

Unix для этого используется девять битов. Три из них позволяют владельцу по отдельности задавать доступ на чтение, доступ на запись и доступ на выполнение. В случае обычных текстовых файлов у владельцев обычно есть доступ на чтение и запись, а доступ на выполнение возможен только для исполняемых программ или сценариев оболочки. Еще три бита выделены для группы владельца, что позволяет устанавливать разграничения, например, между студентами и факультетом. Последние три бита относятся ко всем остальным пользователям.

Этот механизм значительно проще существовавшего в Multics, и он используется уже долгое время. Например, стандартные команды, такие как редакторы, компиляторы, оболочки и т. п., принадлежат привилегированной учетной записи, обычно пользователю `root`, который может читать, записывать и выполнять их по своему усмотрению, в то время как обычные пользователи могут запускать их (и возможно, читать), но не имеют доступа на запись. Обратите внимание, что выполнить программу можно, не имея возможности ее прочитать; такой подход позволяет защитить содержимое программы.

Достаточно рано появилось такое дополнение, как десятый бит, называемый *setuid* (set user id — установка ID пользователя). Установка этого флага разрешает пользователям запускать исполняемые файлы с правами владельца или группы такого файла. В этом случае программа автоматически меняет идентификатор пользователя на идентификатор владельца файла. Это применяется в программах, которые манипулируют файловой системой, выполняя создание каталогов, переименование файлов и другие операции подобного рода. Программы, осуществляющие привилегированные системные вызовы, принадлежат суперпользователю с неограниченным доступом. Естественно, все программы с этим флагом должны быть тщательно спроектированы, чтобы работа с ними не создавала угрозу безопасности системы. Бит setuid изобрел Деннис Ритчи и в 1979 году получил на него патент.

Как я уже упоминал, идея паролей появилась в операционной системе CTSS, откуда перетекла в Multics, а затем и в Unix. Текстовый файл /etc/passwd содержит строку для каждого пользователя с именем для входа в систему, номером пользователя, паролем и несколькими другими полями. С давних времен пароль хранился в хешированной форме. Хешированием называется форма шифрования, при которой воссоздать оригинал можно

только путем полного перебора. Это означает, что прочитать файл паролей может кто угодно, но использовать хешированные пароли для входа в систему под именем другого пользователя невозможно.

Впрочем, это в теории. При не очень хорошем хешировании или плохих паролях расшифровка возможна. В середине 1970-х Кен и Боб Моррис собирали файлы паролей из различных систем Unix и экспериментировали со словарными атаками, проверяя, хэшируются ли возможные пароли в вариант, сохраненный в файле паролей. Эксперименты показали, что таким способом можно получить от 10 до 30 % паролей.

Словарные атаки все еще эффективны, несмотря на всестороннее усложнение технологий. Казалось бы, современные пользователи лучше должны быть осведомлены об опасности слабых паролей, но недавние списки часто используемых паролей показали, что это не так. Перебором по словарю пользовался и «Червь Морриса», которого в 1998 году непреднамеренно выпустил сын Боба Роберт Т. Моррис. Этот червь пытался войти в системы Unix через интернет и распространить себя дальше. Один из его механизмов применял словарь вероятных паролей, таких как «password» или «12345».

Оригинальную команду `crypt` тоже написал Боб. В 1986 году он ушел из Bell Labs, чтобы стать главным научным сотрудником Агентства национальной безопасности (NSA), что говорит о том, как глубоко он разбирался в компьютерной безопасности и криптографии. Умер Боб в 2011 году в возрасте 78 лет.

Криптография была предметом постоянного интереса нескольких сотрудников подразделения 1127, включая Боба, Кена, Денниса, Питера Вайнбергера и Фреда Граммпа (рис. 5.19). На своей веб-странице Деннис рассказывает некоторые интересные истории. Сегодня шифрование выполняется с помощью программного обеспечения, а во время Второй мировой войны для этого применялись механические устройства. Фред каким-то образом раздобыл переносную шифровальную машину «Энигма», которой пользовались немецкие военные. Мне доводилось слышать две версии: что он сам купил ее на рынке и что ее привез из Германии после окончания войны его отец. Фред хранил ее в Bell Labs, а после его смерти она перешла к Кену Томпсону, который держал ее в своем кабинете в нижнем ящике шкафа для бумаг.

Однажды я одолжил ее для лекции по криптографии, которую читал в Принстоне. Как оказалось, никто из моих студентов никогда не видел «Энигму». Когда я водрузил ее на кафедру, многие забрались на столы, чтобы лучше рассмотреть. Никогда ранее я не видел у студентов такого интереса. Впоследствии Кен подарил «Энигму» музею.

Рис. 5.19. Фред Грампп, примерно 1981 год (из архива Джерарда Хольцманна)

Когда в 1983 году Кен и Деннис получили премию Тьюринга, в своем пророческом выступлении «Размышления о доверии к доверию» Кен рассказал, какие изменения в компиляторе позволят внедрить троян в программу входа в систему.

Вы не можете доверять никакому коду, если только вы не написали его сами, особенно нельзя доверять «коду от компаний, которые берут на работу таких, как я». Никакая проверка на уровне кода или тщательное изучение не защитят вас от ненадежного кода.

Он отметил, что аналогичные приемы можно применять к аппаратным средствам, где их еще труднее обнаружить. Со временем ситуация не улучшилась, этот документ актуален и сегодня.

АППАРАТНОЕ ОБЕСПЕЧЕНИЕ

Деятельность подразделения 1127 была сосредоточена в основном на программном обеспечении, но оборудование нас тоже весьма интересовало. Изначально компетентность в сфере аппаратного обеспечения требовалась для подключения к компьютеру PDP-11 дополнительной периферии, например синтезатора голоса Votrax, телефонного оборудования, наборных устройств и различных сетевых устройств. Постепенно был разработан набор инструментов для автоматизированного проектирования (САПР). В разработке принимало участие множество людей, в том числе Джо Кондон, Ли Макмэхон, Барт Локанти, Сэнди Фрейзер, Эндрю Хьюм и др.

В начале 1980-х с помощью инструментов САПР Барт спроектировал и создал растровый терминал. В то время большинство терминалов могли отображать только 24 строки по 80 символов ASCII фиксированной ширины и фиксированной высоты. Растровый терминал позволял отображать большой массив пикселей, значения каждого из которых устанавливались независимо, как на современных экранах ноутбуков и сотовых телефонов. Впрочем, первые растровые изображения были монохромными. Первоначально Барт назвал свой терминал Jerq, намекая на компьютерную рабочую станцию Perq, выпускавшуюся компанией Three Rivers.

Терминал Jerq начал свое существование вместе с очень популярным в то время микропроцессором Motorola 68000 (например, его часто использовали на рабочих станциях), но и его название, и реализацию пришлось принести в жертву корпоративной политике. Он был переименован в Blit (намек на операцию bitblit, быстро обновляющую содержимое экрана) и выпускался небольшими партиями. Компания Western Electric (производственное подразделение AT&T) модернизировала его под процессор Bellmac 32000 — еще одну выпускаемую ими разработку Bell Labs. Новая модель называлась DMD-5620. Эти переделки заняли целый год, и AT&T потеряла возможность конкурировать на растущем рынке рабочих станций.

Большую часть ОС для Blit и 5620 написал Роб Пайк. Самым инновационным аспектом в его работе стали вычисления в перекрывающихся окнах. Перекрывающиеся окна отображались на экране и раньше, но при этом активным было только одно из них. Роб получил патент на это усовершенствование.

Терминал 5620 получился хорошим, хотя тяжелым и громоздким. Я использовал его для написания графических программ, таких как средство предварительного просмотра для Troff. А Роб Пайк написал для этого терминала серию текстовых редакторов, в которых можно было использовать мышь. Одним из них я пользуюсь до сих пор: эта книга была написана в редакторе Sam.

Наблюдался также устойчивый интерес к интегральным микросхемам и СБИС (сверхбольшим интегральным схемам). В 1980 году подразделению 1127 предложили трехнедельный ускоренный курс по проектированию интегральных микросхем, который вел Карвер Мид из Калифорнийского технологического института. Линн Конвей и Карвер написали книгу (*Introduction to VLSI Systems*, 1980) о способах проектирования и внедрения интегральных микросхем, и их курс уже читался в ряде университетов. У Карвера был дар рассказывать о принципах работы схем простым языком, иллюстрируя повествование отвлеченными примерами.

В результате превосходной подачи материала через несколько недель все в классе Карвера научились проектировать и создавать экспериментальные микросхемы. Чипы изготавливались на заводе Bell Labs в Аллентауне, штат Пенсильвания, и возвращались к нам для экспериментов. В то время актуальной была технология 3,5 микрона. Современные схемы обычно имеют толщину слоя от 7 до 10 нанометров, то есть разрешающая способность литографии увеличилась по меньшей мере в 300 раз.

Моя разработка — простые шахматные часы — так и не заработала из-за грубой логической ошибки, которую я осознал намного позднее. Были те, кто разрабатывал не только собственные чипы, но и инструменты поддержки. Моим вкладом стала программа по трассировке проводников на чипе, так что, несмотря на провал в качестве разработчика микросхем, я не считаю эти недели потраченными зря.

За эти годы по крайней мере полдюжины человек в моем отделе использовали СБИС в той или иной форме: для алгоритмов проверки макетов, в качестве симуляторов, для инженерного анализа и ряда теоретических исследований. Понимать, что они делали, я мог только благодаря курсу Карвера.

Интерес к СБИС сохранялся длительное время, что в конечном итоге привело к появлению микропроцессора CRISP (C Reduced Instruction Set

Process) Дейва Дитцела и Рэя Маклеллана. Это был один из первых процессоров семейства RISC. Аббревиатура RISC получена из определения Reduced Instruction Set Computer (компьютеры с уменьшенным набором инструкций). Это семейство архитектур процессоров более простое и регулярное, чем такие процессоры, как VAX-11/780.

Микропроцессор CRISP имел набор инструкций, хорошо подходящих для вывода компилятора Си. Для его проектирования Дейв тесно сотрудничал со Стивом Джонсоном. После обсуждения потенциальных возможностей архитектуры Стив модифицировал компилятор Portable C и запускал тесты производительности, чтобы посмотреть, как на нее влияет архитектура процессора. Еще один отличный пример совместного проектирования аппаратного и программного обеспечения.

В конечном итоге AT&T продала версию CRISP под названием Hobbit. Она предназначалась для одного из первых карманных персональных компьютеров Apple Newton, но ни Newton, ни Hobbit не имели особого коммерческого успеха. В 1995 году Дейв покинул Bell Labs, чтобы основать компанию Transmeta, специализирующуюся на процессорах с низким энергопотреблением.

Хотя сам микропроцессор CRISP не имел коммерческого успеха, операционная система Unix и язык Си оказали большое влияние на вычислительную технику 1980-х и 1990-х годов. Наиболее успешные архитектуры набора команд хорошо согласовывались с Си и Unix. Кроссплатформенность Unix и Си позволила университетам и особенно компаниям создавать новые архитектуры и быстро портировать программное обеспечение. При этом возникло требование, чтобы набор инструкций подходил для кода на Си. Функциональные особенности, которые трудно компилировались из программ на Си, при этом стремились убрать. Методология проектирования процессоров, подобная методологии Джонсона и Дитцела, использовала статистический анализ программ, соответственно, внедрялись в основном те вещи, которые обеспечивали быструю работу Си. Важность Unix и Си была так велика, что проектирование процессоров в 1980-х и 1990-х годах вращалось вокруг них. Никто не создал успешных процессоров, настроенных на другие языки.

ГЛАВА 6

ПО ТУ СТОРОНУ ИССЛЕДОВАНИЙ

> В настоящее время операционная система Unix используется в 1400 университетах и колледжах по всему миру. Она легла в основу 70 компьютерных серий от микро- до суперкомпьютеров. Сейчас в эксплуатации находится порядка ста тысяч систем Unix, и около ста компаний разрабатывают приложения на ее основе.
>
> *Р. Л. Мартин, Unix System Readings and Applications, том 2, 1984 год*

После нескольких лет эксплуатации внутри подразделения 1127 операционная система Unix начала распространяться по Bell Labs и за ее пределы, причем в основном через университеты, которые могли получить исходный код, подписав соглашение о коммерческой тайне. Про открытый исходный код речи еще не шло. Системой можно было пользоваться только в образовательных целях, а владельцы лицензий могли обсуждать свой опыт работы с Unix только с другими лицензированными пользователями. Однако сообщество быстро росло, по всему миру появлялись группы пользователей и добавлялись крупные технические инновации, например порты для различного оборудования и новые механизмы доступа к интернету.

Многое, о чем пойдет речь в этой главе, проходило параллельно или даже раньше, чем события из предыдущей главы. Это слегка запутывает хронологию моего повествования.

6.1. ГРУППА PROGRAMMER'S WORKBENCH

Первым заказчиком новой операционной системы стал патентный отдел Bell Labs. Затем Unix сочли полезной и другие группы, и система уже на ранних стадиях своего существования начала распространяться как среди разработчиков из Bell Labs, так и среди других подразделений AT&T.

Насчитывающая более миллиона сотрудников, AT&T была очень крупной компанией с множеством компьютерных систем для управления данными и операций, поддерживающих услуги телефонной связи. Эти системы предоставляли технические интерфейсы и поддержку, ведя учет оборудования и клиентов, контролируя работающие системы, записывая в журнал события, устраняя неисправности и т. п. В совокупности они назывались системами эксплуатационной поддержки.

Одна из первых масштабных установок Unix за пределами отдела исследований была предоставлена в распоряжение группы, базирующейся примерно в 15 милях от Мюррей-Хилл в Пискатауэе, штат Нью-Джерси. В 1973 году эта группа начала проектировать инструменты разработки программного обеспечения для крупного производства. Набор этих инструментов назвали Programmer's Workbench, или PWB.

Большинство систем эксплуатационной поддержки в AT&T работало на больших мэйнфреймах от IBM и Univac со своими собственными закрытыми ОС, такими как IBM OS/360. Группа PWB предоставляла средства создания программного обеспечения, которое можно было устанавливать на такие компьютеры, а также средства управления им. По сути, система PWB Unix служила унифицированным интерфейсом для разнообразного набора крупных компьютерных систем, не принадлежащих семейству Unix. Мейнфреймы рассматривались как периферийные устройства.

Одной из основных служб PWB был удаленный ввод заданий. Это набор команд для отправки заданий в целевые системы и возврата результатов. Еще эти команды отвечали за очереди заданий, отчеты о состоянии, уведомления, ведение журнала и устранение ошибок. Удаленный ввод заданий хорошо сочетается с подходом Unix, в котором практикуется применение небольших инструментов, допускающих разные способы подключения, а также инкапсуляцию в сценарии оболочки, чтобы ими могли без труда воспользоваться даже неспециалисты.

Для поддержки этого вида программирования Джон Мэши (рис. 6.1) внес усовершенствования в оболочку шестой редакции и назвал новую версию оболочки PWB. Эта версия значительно увеличила удобство процесса программирования, так как в ней появился общий метод `if-then-else` для принятия решений, цикл `while` и переменные оболочки для хранения текста. Еще Джон изобрел механизм пути поиска, позволяющий любому пользователю указать через переменную оболочки последовательность каталогов для поиска команд. Это давало возможность группам пользователей работать со своими программами прямо из каталогов проектов без необходимости устанавливать их в системные папки, для записи в которые у них иногда даже не было разрешения. Как сказал сам Джон:

> У нас было множество пользователей (более тысячи), которые обычно не были программистами на Си и работали в группах в общих средах. Они хотели совместно использовать собственные наборы команд в лабораториях, отделах, группах. Они часто работали в системах с совместным доступом, где никто не мог и не должен быть назначен суперпользователем.

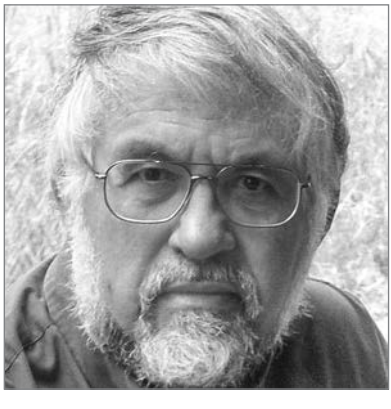


Рис. 6.1. Джон Мэши, примерно 2011 год (фото из Twitter)

Еще Джон добавил механизм, позволяющий при пометке файла как исполняемого либо выполнять его как обычную команду, либо (если это был сценарий) передавать в оболочку. Все эти функции были внедрены к началу 1975 года и усовершенствованы в течение следующего года, поскольку оболочкой PWB стало пользоваться все больше и больше людей. В своей

статье *Using a Command Language as a High-Level Programming Language* Джон поделился опытом применения более 1700 процедур оболочки:

> Командный язык оболочки позволял пользователям PWB избежать трудоемкого программирования, которое часто сопровождает большой проект. Многие процедуры удобно автоматизированы. Легкость создания и использования процедур оболочки дает возможность настраивать среду PWB для каждого проекта, адаптируя ее к индивидуальным требованиям, организационной структуре и терминологии.

Как я отмечал в предыдущей главе, улучшения, внесенные Джоном, вскоре нашли свое отражение в оболочке, написанной Стивом Борном.

Другой важной разработкой группы PWB была первая система управления версиями (Source Code Control System, SCCS), написанная в 1972 году Марком Рочкиндом. Она стала первой из целого набора программ управления большими базами кода, над которым работают несколько человек одновременно.

Основная идея SCCS состояла в том, что программист мог получить фрагмент кода для работы. Эта часть кода блокировалась, и до снятия блокировки у других программистов не было доступа к ее редактированию, что исключало одновременное внесение противоречащих друг другу изменений. Конечно, проблемы тоже возникали. Например, из-за небрежности или сбоя мог блокироваться код, над которым никто не работал. Или при слишком большой заблокированной области падала скорость редактирования других фрагментов.

Но сама идея оказала сильное влияние на разработку ПО в команде. Сегодня она еще более актуальна, потому что над большими базами кода иногда трудится множество разработчиков, живущих в разных местах. Можно четко проследить эволюционный путь от SCCS через RCS, CVS и Subversion к Git — современной повсеместно используемой стандартной системе контроля версий.

Еще Марк Рочкинд изобрел инструмент, преобразующий набор регулярных выражений в программу на Си, которая сканирует системные журналы, проверяя их на вхождения заданных шаблонов, и выводит сообщение при обнаружении соответствий. Идея оказалась настолько изящной, что Аль

Ахо, Питер Вайнбергер и я ~~украли~~ адаптировали и обобщили ее для модели шаблон-действие в языке Awk.

В версию PWB также входил пакет Writer's Workbench (WWB), помогающий людям писать более качественные тексты. Джон Мэши и Дейл Смит при поддержке Теда Долотта создали набор универсальных команд Troff, назвав полученный пакет Memorandum Macro, или `mm`. Он широко применялся для производства документации как в компании AT&T, так и в других местах.

В пакет WWB в числе прочего входят проверка орфографии, программы поиска пунктуационных ошибок, разделения инфинитивов и повторяющихся слов (подобные вещи часто появляются в результате редактирования). Еще он включает инструменты проверки грамматики и стиля, а также оценки читабельности. Основным компонентом стала программа Лоринды Черри `parts`, определяющая, к какой части речи относятся слова текста. Работала программа неидеально, но позволяла получить статистические данные о частоте прилагательных, сложных предложений и прочего. Пакет WWB появился в конце 1970-х годов, когда авторы все чаще стали использовать компьютеры. Он получил хорошие отзывы в прессе, а двое из его разработчиков, Лоринда Черри и Нина Макдональд, даже выступали в шоу *Today* на канале NBC.

Чтобы показать, насколько за прошедшие годы упала цена и увеличилась мощность вычислительного оборудования, процитирую фрагмент статьи Теда Долотта и Мэши, опубликованной в 1978 году, где описывалась среда разработки, поддерживающая более тысячи пользователей: «По совокупности показателей это самая большая из известных систем под управлением Unix». Она работала в сети из семи машин PDP-11 с *общим* объемом оперативной памяти 3,3 Мбайт и 2 Гбайт дискового пространства. Это примерно одна тысячная часть типичного современного ноутбука сегодня. Сможет ли на вашем ноутбуке работать миллион пользователей?

6.2. ЛИЦЕНЗИИ ДЛЯ УНИВЕРСИТЕТОВ

В 1973 году AT&T за номинальную плату начала продавать лицензии на Unix университетам. По большей части это были лицензии на 6-ю редакцию, которая вышла в 1975 году. Было и несколько коммерческих лицензий

стоимостью 20 тысяч долларов. В современном эквиваленте эта сумма превышает 100 тысяч долларов. Лицензия давала возможность получить исходный код ОС, но не поддержку.

Одним из наиболее активных получателей лицензий стал Калифорнийский университет в Беркли. Его аспиранты внесли большой вклад в систему распространения программного обеспечения в исходных кодах (Berkeley Software Distribution, BSD) и в BSD-UNIX — одну из двух основных ветвей, возникших на базе исходных версий, выходящих в Research Unix.

В 1975 и 1976 годах Кен Томпсон проводил творческий отпуск в Беркли, где читал курсы по операционным системам. Билл Джой (рис. 6.2), в то время учившийся в аспирантуре, добавил в локальную версию Unix несколько собственных программ, в том числе текстовый редактор `vi`, который до сих пор остается одним из самых популярных редакторов Unix, и написанную на Си оболочку csh. Позже Билл разработал для Unix сетевую модель передачи данных TCP/IP, используемую и по сей день. Придуманный им программный интерфейс `socket` позволял читать и записывать сетевые соединения с системными вызовами `read` и `write`, применявшиеся для файлов и устройств ввода-вывода, что упростило развитие сетевой функциональности.

В середине и конце 1970-х Билл время от времени посещал Bell Labs. Я помню, как он показывал мне новый текстовый редактор, над которым тогда работал. К этому времени печатающие терминалы были вытеснены терминалами на основе электронно-лучевой трубки, что сделало процесс редактирования намного более интерактивным.

В `ed` и других редакторах того времени набирались команды, меняющие текст, но чтобы посмотреть на результат их действия, требовалось в явном виде вывести новую строку. Например, в `ed` команда

```
s/this/that/p
```

меняла в текущей строке `this` на `that` и отображала результат. Были и команды, позволяющие выполнять замену в разных строках одновременно, искать целые строки, выводить диапазоны строк и т. п. Новичку было непросто разобраться в `ed`, но в руках специалиста он был весьма эффективным инструментом.

Рис. 6.2. Билл Джой, примерно 1980 год (фото из архива Билла Джоя)

В редакторе Билла для обновления экрана в процессе редактирования текста применялось позиционирование курсора. После построчной модели это было серьезное изменение. Курсор помещался на слово `this` (например, с помощью регулярного выражения), набиралась команда `cw` (change word), а затем слово `that`, которое сразу же замещало оригинал.

Не помню, что я тогда говорил о самом редакторе (хотя сегодня `vi` — один из двух редакторов, которыми я пользуюсь чаще всего), но помню, как советовал Биллу прекратить возню с редакторами и дописать докторскую диссертацию. К счастью для него и многих других, мой совет он проигнорировал. Несколько лет спустя он бросил учебу в аспирантуре и стал одним из первых разработчиков рабочих станций Sun Microsystems с программ-

ным обеспечением на базе дистрибутивов Университета Беркли, которые включали результаты его фундаментальной работы над системой, сетями и инструментами (и его редактор `vi`). Эту историю я часто рассказываю студентам, когда у меня просят совета по поводу профориентации. Старший не всегда означает более мудрый.

6.3. ПОЛЬЗОВАТЕЛЬСКИЕ ГРУППЫ И USENIX

Поскольку AT&T не предоставляла владельцам лицензий Unix никакой поддержки, пользователям приходилось объединяться для помощи друг другу, что в итоге привело к регулярным встречам с техническими презентациями, обменом программным обеспечением и, конечно же, общественной деятельностью. Разумеется, пользователи Unix не были первооткрывателями в этой сфере. Например, группа SHARE для корпоративных пользователей мэйнфреймов IBM появилась в 1955 году и по-прежнему активна. Аналогичные сообщества формировались и для других производителей оборудования.

Первое собрание группы пользователей Unix состоялось в 1974 году в Нью-Йорке. Постепенно эти группы стали появляться по всему миру. В 1979-м мы с Кеном посетили первое собрание группы UKUUG в Кентском университете в Кентербери. Для меня этот первый опыт посещения Англии стал настоящим приключением. Мы прилетели в аэропорт Гатвик на самолете Laker Airways — первом из трансатлантических лоукостеров. Поехали в Солсбери, чтобы посмотреть Солсберийский собор и Стоунхендж, а затем отправились в Кентербери на встречу (и увидеть Кентерберийский собор). После этого я несколько дней восторженно гулял по Лондону.

С тех пор под предлогом встреч с группами пользователей Unix я побывал в нескольких странах. Эти встречи — отличный способ познакомиться с очень хорошими людьми. Самым запоминающимся стало путешествие в Австралию в 1984 году. Мы с Кеном полетели на встречу, которая проходила в Сиднейском оперном театре. В первый день я выступал с докладом, а остаток недели провел, смотря на гавань из окна конференц-зала. Это было настолько увлекательно, что я даже не помню, о чем говорилось на встрече.

Группы пользователей превратились в ассоциацию «Группы пользователей Unix», которую после жалобы AT&T на незаконное использование товарной марки Unix переименовали в USENIX (далее Usenix). Сегодня Usenix проводит множество профессиональных конференций и выпускает технический журнал *;login:*. Ассоциация Usenix сыграла значительную роль в распространении Unix, выпуская презентации и учебные пособия. Кроме того, она способствовала развитию технологии UUCP и поддерживала новостные группы Usenet.

6.4. КНИГА ЛАЙОНСА

Джон Лайонс (рис. 6.3), профессор computer science из Университета Нового Южного Уэльса (UNSW) в Сиднее, достаточно рано стал приверженцем Unix и широко пользовался ею на курсах по операционным системам, а также в образовательных и исследовательских целях.

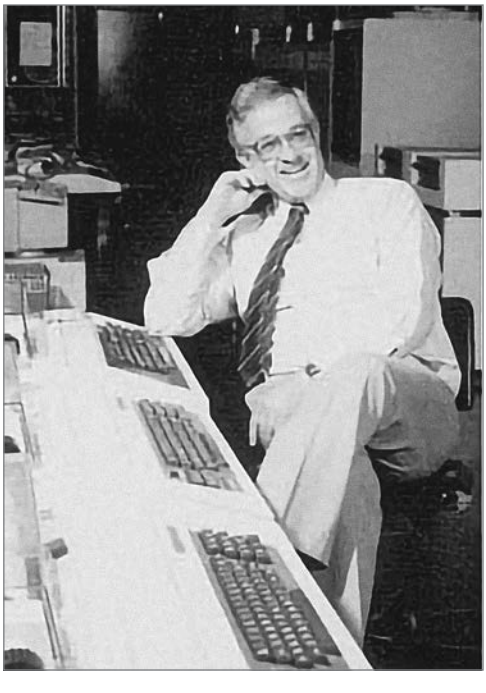


Рис. 6.3. Джон Лайонс (из архива Университета Нового Южного Уэльса)

В 1977 году Джон написал построчные комментарии к исходному коду 6-й редакции. Каждый фрагмент исходного кода получил подробные объяснения. Любой мог понять, как это работает, почему именно так и как это можно сделать по-другому. Джон воспитал отличных студентов, несколько из них потом работали в Bell Labs.

Оригинальный выпуск «Комментариев» был в двух томах: в одном — код, в другом — толкования. Их можно было читать параллельно, а вот официальная версия (рис. 6.4), вышедшая в 1996 году, была уже однотомной.

Несмотря на возможность распространять книгу среди владельцев лицензии на Unix, с технической точки зрения она составляла коммерческую тайну, поскольку опубликованный в ней код был частной собственностью правообладателя — компании AT&T. Копии тщательно контролировались, по крайней мере в теории. У меня до сих пор хранится мой нумерованный экземпляр (№ 135). Но сразу после своего появления книга начала широко копироваться. Кроме того, самиздатовская обложка с рис. 6.4 показывает, что копии печатались негласно. Спустя годы, когда уже невозможно стало игнорировать реальность, книга Джона поступила в продажу.

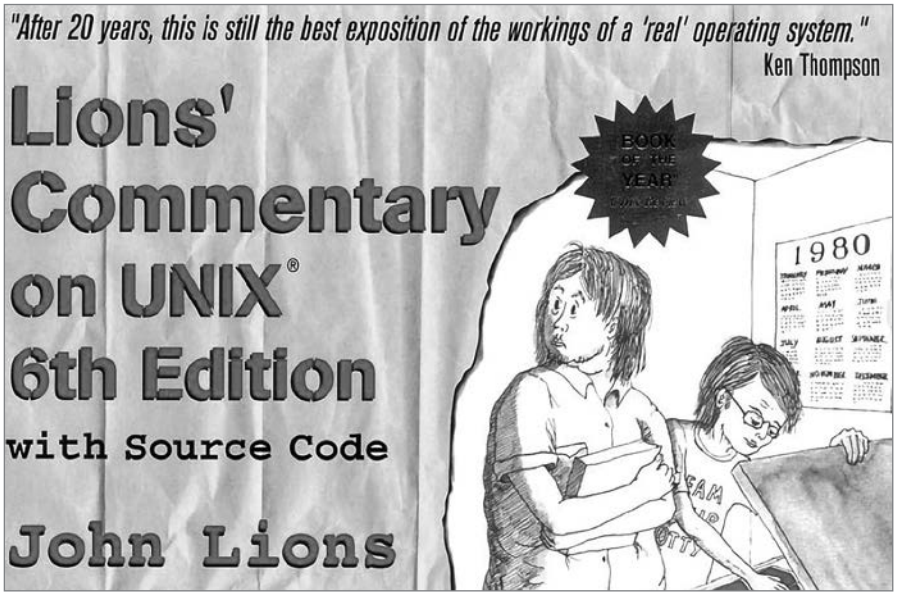


Рис. 6.4. Комментарии Джона Лайонса к 6-й редакции Unix

В 1978 году Джон провел творческий отпуск в Мюррей-Хилл, в кабинете Денниса Ритчи, который располагался напротив моего. Умер Джон в 1998 году в возрасте 62 лет. В память о его работе в UNSW создали кафедру операционных систем. Эта кафедра спонсировалась выпускниками и друзьями Джона. Тед Долотта пожертвовал туда средства, вырученные в 1998 году от продажи на аукционе своего калифорнийского автомобильного номера с номерным знаком UNIX, который был приобретен Джоном Мэши.

Благодаря «Комментариям Лайонса» приобрел известность один комментарий в исходном коде Unix. Строка 2238 гласит:

```
/* От вас не ожидают, что вы это поймете. */
```

Как я уже упоминал, Деннис умер в октябре 2011 года. Этот комментарий я взял как основу для своей речи на вечере его памяти, который состоялся в Bell Labs год спустя.

Код ядра Unix Деннис и Кен Томпсон писали вместе. Насколько я знаю, Кен всегда считал, что хороший код не требует много комментариев. Экстраполируя эту идею, можно сказать, что отличный код в комментариях вообще не нуждается. Поэтому думаю, что большинство комментариев в ядре написаны Деннисом. Комментарий в строке 2238 приобрел известность благодаря завуалированному сарказму. Много лет он появлялся на футболках и других вещах. Как сказал сам Деннис:

> Его часто цитируют как высказывание, бросающее тень на количество или качество комментариев в экспериментальных версиях Unix, которые выходили в Bell Labs. Боюсь, что это в целом справедливое замечание в данном случае неоправданно.

Если вы внимательно прочитаете тот код, то увидите, что обсуждаемый комментарий следует за более длинным, описывающим механизм переключения контекста, при котором останавливается выполнение одного процесса и начинается выполнение другого. И это попытка объяснить действительно сложную материю. Снова дам слово Деннису:

> Фраза «от вас не ожидают, что вы это поймете» не была дерзкой провокацией, она подразумевала всего лишь, что этого не будут спрашивать на экзамене.

Ранее я упоминал, насколько сложными в освоении были инструменты Nroff и Troff. Последний абзац раздела «Благодарности» в «Комментариях Лайонса» наводит на мысль, что Джон разделяет это мнение:

> Следует также отметить содействие программы nroff. Без него эти заметки никогда не появились бы в таком виде. Но некоторые свои секреты эта программа открывала так неохотно, что к благодарности автора примешиваются и другие чувства. Определенно nroff должен стать благодатной почвой для тех, кто решит практиковать искусство документирования программ.

6.5. ПОРТИРУЕМОСТЬ

Шестая редакция Unix в основном написана на Си с небольшими вкраплениями ассемблера для доступа к функциям компьютера, недоступным другим способом, например к настройке регистров, отображению памяти и т. п. В это время Стив Джонсон создал «переносимую» версию компилятора Си, которую можно было настроить на генерацию языка ассемблера для архитектур, отличных от PDP-11. Благодаря этому компилятору появилась возможность переносить программы на Си на другие виды компьютеров, просто перекомпилировав их. Очевидно, что наиболее интересной для этой операции программой была сама ОС.

Первый перенос Unix осуществил Ричард Миллер в австралийском Университете Вуллонгонг в Новом Южном Уэльсе. Перенос был выполнен на компьютер Interdata 7/32. Миллер не пользовался переносимой версией компилятора Си, а отредактировал оригинальный компилятор Денниса Ритчи. К апрелю 1977 года он получил собственную версию Unix.

Независимо от Миллера, Стив Джонсон и Деннис Ритчи портировали Unix на похожую машину, Interdata 8/32. Правда, они преследовали несколько иную цель: получить более переносимую версию Unix, а не просто отдельную копию оригинальной системы. Свою версию они выпустили в конце 1977 года. Стив Джонсон вспоминает:

Была еще одна веская причина сделать Unix портируемой. Некоторые конкуренты DEC начинали жаловаться на слишком тесные взаимоотношения AT&T и DEC. Отсутствие на рынке альтернативы PDP-11 не воспринималось как аргумент. Деннис подключил меня к работе над переносимостью одной фразой: «Думаю, что проще перенести Unix на другую платформу, чем переписывать приложение под другую операционную систему». С этого момента я был в деле.

Переносимость стала большим шагом вперед. До этого момента операционные системы в основном писались на ассемблере, и даже версии, написанные на языке высокого уровня, были в той или иной степени привязаны к конкретной архитектуре. Но после проектов Миллера, Джонсона и Ритчи перенос Unix на другие типы компьютеров при всей его нетривиальности уже не вызывал затруднений. Это серьезно повлияло на развивающийся рынок рабочих станций, на котором старые и новые компании создавали компьютеры меньше и дешевле таких мини-компьютеров, как PDP-11 и Interdata, и использовали разные процессоры.

Рабочие станции предполагались в качестве персональных компьютеров для ученых и инженеров как мощная и обычно неразделенная вычислительная среда. Наиболее коммерчески успешными оказались рабочие станции от Sun Microsystems. Еще их производили компании Silicon Graphics, DEC, Hewlett-Packard, NeXT и даже IBM. У первых рабочих станций в начале 1980-х предполагался как минимум мегабайт оперативной памяти, мегапиксельный дисплей и вычислительная мощность в миллион инструкций в секунду. Эта концепция называлась компьютером «3М». Для сравнения: мой видавший виды Macbook имеет 8 Гбайт оперативной памяти и скорость не менее миллиарда операций в секунду. Его дисплей чуть больше мегапикселя, но пиксели не монохромны, а имеют 24-битный цвет.

Рынок рабочих станций возник благодаря технологическим усовершенствованиям, которые позволили упаковать серьезные вычислительные мощности в небольшой блок и продавать его по скромной цене. Полная цена системы стала приемлемой отчасти по причине доступности программного обеспечения, в том числе операционной системы. Новым производителям

уже не требовалось создавать операционную систему с нуля, достаточно было портировать Unix и сопутствующие программы под используемый процессор. Таким образом, развитию рынка рабочих станций сильно способствовала доступность Unix.

Первый персональный компьютер IBM (ПК) появился в 1981 году. Он и его многочисленные клоны были в 5–10 раз дешевле рабочей станции. По своим характеристикам они изначально были совсем не конкурентоспособными, но постепенно ситуация улучшалась, и к концу 1990-х эти характеристики выровнялись. Разница между рабочей станцией и ПК постепенно стерлась. Сегодня в зависимости от областей применения такие машины чаще всего работают под управлением Microsoft Windows, macOS, Unix или Unix-подобных систем.

ГЛАВА 7

КОММЕРЧЕСКАЯ РЕАЛИЗАЦИЯ

По мере того как операционная система Unix распространялась в академических кругах, о ней узнавали и компании, ведь недавно нанятые ими программисты пользовались ей во время обучения.

С сайта компании Lucent, 2002 год

Unix и Си — совершенные компьютерные вирусы.

Ричард Гэбриел, «Чем хуже, тем лучше», 1991 год

Считалось, что компании AT&T запретили коммерческое распространение Unix, потому что как государственная монополия она составляла конкуренцию другим поставщикам операционных систем, используя доходы от телекоммуникационных услуг для перекрестного субсидирования разработки Unix. Самое большее, что AT&T смогла сделать в этой ситуации, — продавать лицензии Unix корпоративным клиентам за 20 тысяч долларов (в то время как образовательные учреждения получали лицензию на льготных условиях), но в ограниченных количествах и без какой-либо поддержки. Такая политика позволила избежать санкций от регулирующих органов.

7.1. РАЗДЕЛЕНИЕ

Несмотря на регуляционные меры, к 1980 году начались нападки на AT&T как на монополию. Еще в 1974 году Министерство юстиции США начало антимонопольный процесс против AT&T на том основании, что компания обязывала абонентов использовать только телефоны, произведенные Western Electric, которые не продавались, а сдавались в аренду, и арендная плата составляла значительную часть доходов. Министерство юстиции потребовало от AT&T отделить подразделение Western Electric, которое производило оборудование.

Вместо этого AT&T предложила разделение на основной филиал, предоставляющий услуги междугородной связи, и семь региональных телефонных компаний (Baby Bells), которые предоставляли бы местную телефонную связь в своих географических зонах. Подразделение Western Electric AT&T хотела сохранить за собой, согласившись убрать требование к операционным компаниям пользоваться только их оборудованием. Оставляла она себе и Bell Labs.

Мировое соглашение с Министерством юстиции, по которому AT&T отказалась от компаний-операторов, было окончательно достигнуто в начале 1982 года и вступило в силу 1 января 1984 года.

Разделение стало резким поворотным моментом, который в конечном итоге привел к упадку AT&T. Следующие 20 лет просчетов и неправильных решений превратили Bell Labs в жалкое подобие того, что было раньше, когда у них была ясная и четкая миссия, а также адекватное и стабильное финансирование.

В 1984 году от Bell Labs отделилась дочерняя копания, получившая название Bellcore (Bell Communications Research), которая должна была предоставлять исследовательские услуги для Baby Bells. В Bellcore ушло немало людей из отдела исследований, в основном те, кто занимался коммуникациями. Оказались в их числе и некоторые коллеги из подразделения 1127, в частности Майк Леск и Стю Фельдман. Но в какой-то момент региональные телефонные компании решили, что исследования им не требуются, и Bellcore была приобретена компанией SAIC и переименована в Telcordia. В конечном итоге она оказалась в собственности шведской телекоммуникационной компании Ericsson.

Кроме этого, в 1984 году Bell Labs переименовали в AT&T Bell Laboratories, поскольку в соответствии с мировым соглашением во всех прочих случаях AT&T не разрешалось использовать имя Bell. Нам настоятельно рекомендовалось всегда использовать только полное название.

7.2. USL и SVR4

После разделения на смену неспособности и нежеланию AT&T торговать Unix пришли попытки активной коммерческой деятельности, которые начала предпринимать часть компании, организационно весьма удаленная от отдела исследований. Физически они тоже находились от нас довольно далеко, занимая здание в городе Саммит, Нью-Джерси. Так как его окружали оживленные шоссе, неформально его называли Островом автострад (Freeway Island). Первоначально эта организация называлась Группой поддержки Unix (Unix Support Group, USG), но в итоге получила имя Unix System Laboratories, или USL. Первую USG создал в 1973 году Берк Таг в попытке обеспечить централизованную поддержку. Со временем эти группы расширили свою деятельность, в числе прочего занявшись продажами и рекламой.

Несомненно, рынок для продаж Unix существовал. Можно даже сказать, что его непреднамеренно создала AT&T, раздавая лицензии университетам. Ведь студенты, привыкшие работать в этой ОС, хотели продолжать и после выпуска, уже устроившись в коммерческие компании, которые могли позволить себе покупать лицензии за деньги. С 1984 года USL прилагала массу усилий для превращения Unix в профессиональный коммерческий продукт. Кульминацией стала версия System V Release 4, или SVR4. Компания AT&T инвестировала значительные ресурсы в то, чтобы сделать эту версию стандартной реализацией и тщательно определить совместимость как кода, так и объектных модулей. Я думаю, важным было именно внимание к стандартам и совместимости.

Подробности развития SVR4 и взаимодействия AT&T с сотрудниками и конкурентами в течение десяти лет не слишком интересны. Поэтому я не буду на них останавливаться, тем более что в некотором смысле они уже не актуальны: фокус внимания сместился в сторону ОС Linux. Посвященная System V статья в Википедии описывает ситуацию так:

Отраслевые аналитики обычно характеризуют коммерческую версию Unix как переживающую период медленного, но неуклонного упадка.

Разумеется, речь идет только о коммерческой версии; варианты с открытым исходным кодом, такие как представители семейства BSD, о которых мы поговорим в следующей главе, живут и процветают.

В линейку продуктов AT&T кроме операционной системы входили и различные вспомогательные программы, в том числе компиляторы для Си, C++, Фортрана, языка Ada и даже Паскаля. В основном они базировались на портативном компиляторе Си Стива Джонсона. Много сил тратилось на стандартизацию с целью обеспечения совместимости исходного кода и двоичных форматов в библиотеках.

Я в этот период возглавлял отдел Бьёрна Страуструпа, что означало частые контакты с USL по поводу развития C++. По большей части они оказывались взаимовыгодными, но бывали и случаи, когда четко проявлялась разница приоритетов у отдела исследований и организации, отвечавшей за продажу продукта. Например, в 1988 году у меня была бурная дискуссия с менеджером USL.

Менеджер: Нужно исправить все ошибки в компиляторе C++, но его поведение при этом должно остаться без изменений.

Я: Это невозможно. Исправление ошибки по определению приводит к изменению поведения.

Менеджер: Брайан, ты не понял. Ты должен исправить ошибку, но поведение компилятора при этом меняться не может.

Формально я был совершенно прав, но при этом прекрасно понимал, почему на меня давит менеджер: слишком крупные или слишком быстрые изменения — серьезная проблема для тех, кто занимается разработкой программного обеспечения с использованием новых языков и инструментов.

USL открыла в Японии дочернюю компанию Unix Pacific, руководителем которой стал много лет проработавший в Bell Labs в отделе исследований Ларри Крум. В результате в рамках технического сотрудничества я дважды посещал Японию на деньги компании. Во время поездки для обмена опы-

том с крупнейшей телефонной компанией Японии NTT я получил четкое представление о неофициальной иерархии. Исполнительный директор должен был играть в гольф со своим коллегой из NTT. Директор Центра играл в теннис со своим коллегой. Таким, как я, скромным руководителям отделов была предложена поездка за покупками в Токио, от которой я с благодарностью отказался.

Не все попытки AT&T извлечь из Unix коммерческую выгоду оказывались успешными, но стандартизация Unix стала большим подарком всему сообществу. Несмотря на возникавшие время от времени противоречия между отделом исследований и USL, должен сказать, что в USL работала большая группа талантливых коллег, которые внесли значительный вклад в Unix и связанные с ней системы программного обеспечения.

7.3. UNIX™

Когда-то в начале существования Unix ее опекуны из Bell Labs решили, что имя — имеющий ценность товарный знак, который должен охраняться, что с коммерческой точки зрения, безусловно, стало правильным решением. Они пытались не допустить превращения этого имени в общее понятие, которым может пользоваться кто угодно. Подобное уже произошло с такими словами, как аспирин (в США, хотя и не везде), эскалатор, застежка-молния и (относительно недавно) App Store.

В результате появилось требование к сотрудникам Bell Labs: использовать имя корректно. В частности, его нельзя было употреблять как самостоятельное существительное («Unix — это операционная система»). Оно должно идентифицироваться как товарный знак и фигурировать в качестве определения, записанного в верхнем регистре. То есть допустимым был только вариант «операционная система UNIX™», что порождало нелепые предложения вида «Операционная система UNIX™ — это операционная система». В 1984-м нам с Робом Пайком пришлось отстаивать название нашей книги *The Unix Programming Environment* (в русском переводе «UNIX. Программное окружение»), потому что его пытались превратить в *The UNIX™ Operating System Programming Environment*. В конечном итоге мы пришли к компромиссу: на обложке дополнительной информации и товарного знака не будет, но на титульном листе появятся почти невидимая звездочка и примечание.

Громоздкая формулировка была проблемой, особенно для тех, кто серьезно относился к своим текстам, поэтому искались различные обходные пути. Например, в стандартный пакет макросов для Troff `ms` Майк Леск добавил команду форматирования, которая добавляла «UNIX» в верхнем регистре и автоматически создавала сноску на первой странице. Обычно сноска выглядела так:

```
† UNIX is a trademark of Bell Laboratories (UNIX — торговая марка Bell Labs).
```

Но стоило воспользоваться командой с дополнительным недокументированным параметром, как текст менялся:

```
† UNIX is a footnote of Bell Laboratories[1].
```

Не думаю, что когда мы время от времени использовали эту шутку-сюрприз, кто-то хоть раз обращал на нее внимание, но этот код все еще присутствует в стандартном пакете макросов.

При этом слово Unix применялось для товаров и услуг, не имевших ничего общего с операционными системами, например для ручек, показанных на рис. 7.1, для книжных шкафов с рис. 7.2 и огнетушителя с рис. 7.3. Все это, судя по всему, было произведено за пределами США и, соответственно, не попадало под американский закон о товарных знаках. Книжные шкафы так вообще датируются 1941 годом, то есть появились еще до рождения Кена и Денниса. Еще один очаровательный пример: детские подгузники Unix от компании Drypers, которая использовала Unix как сокращение слова unisex (для детей обоего пола).

Рис. 7.1. Ручки Unix (фото предоставлено Арнольдом Роббинсом)

[1] Footnote — не только «сноска», «примечание», но и «мелочь», «нечто, имеющее малое значение». Таким образом, «UNIX — это безделица от Bell Labs». — *Примеч. ред.*

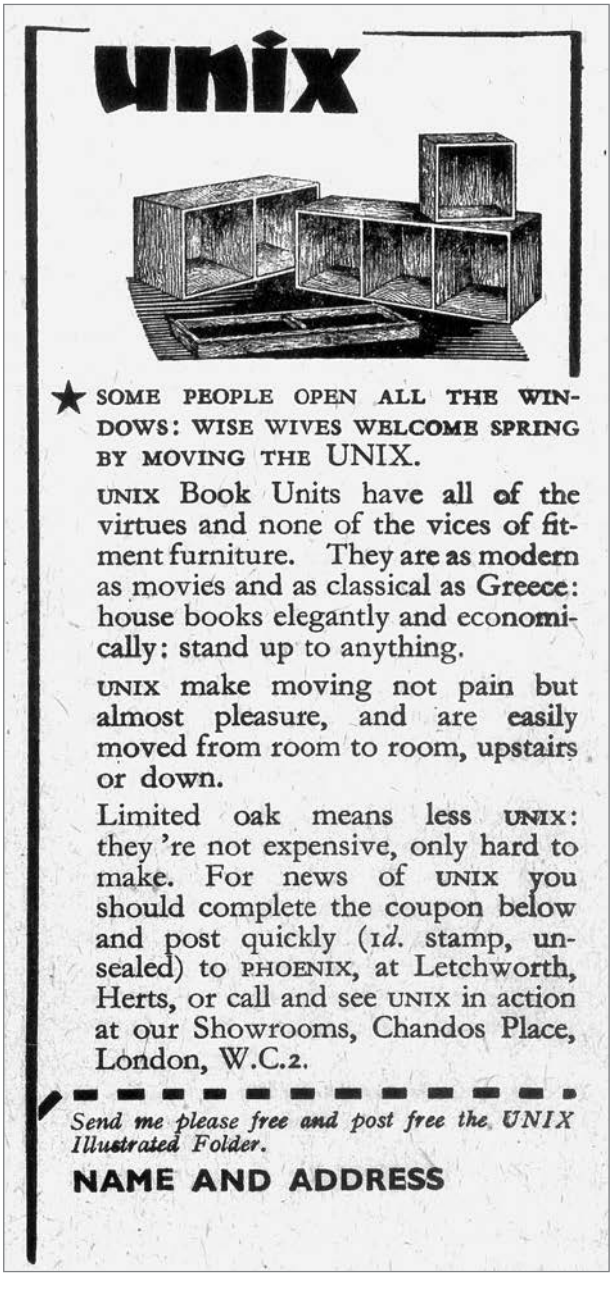


Рис. 7.2. Секционные книжные шкафы Unix, 1941 год (фото предоставлено Иеном Аттингом)

Рис. 7.3. Огнетушитель Unix

7.4. СВЯЗИ С ОБЩЕСТВЕННОСТЬЮ

В Bell Labs постоянно приходили посетители, а с середины 1970-х до начала 1980-х стали проводиться презентации для туристов. Небольшая группа усаживалась в конференц-зале, а члены Центра кратко рассказывали, что такое Unix и почему она важна для AT&T и всего мира. Чаще всего эти презентации проводили Майк Леск и я. Мы обладали одним и тем же недостатком: постоянно на это жаловались, хотя на самом деле нам нравилось.

Среди посетителей встречались как просте смертные, так и «выдающиеся». К последним относились важные для AT&T лица, на которых требовалось произвести впечатление, а иногда и просто обладатели громких имен. Например, в 1980 году я делал презентацию для Уолтера Анненберга, осно-

вателя журнала *TV Guide*. Именно на этом он заработал деньги, которые, возможно, помогли ему стать послом в Великобритании, хотя на момент, когда я показывал ему чудеса Unix, его дипломатическая карьера уже завершилась. Как очень важного гостя его сопровождал президент Bell Labs Билл Бейкер. Я часто включал в свой репертуар демонстрацию конвейеров, показывая, как соединением программ друг с другом можно быстро решать *актуальные* задачи. Я использовал сценарий оболочки для поиска в документе орфографических ошибок, потому что это был хороший пример длинного конвейера, помогающий понять, как существующие программы могут комбинироваться новыми способами.

Сценарий проверки правописания `spell` был создан Стивом Джонсоном. Основная идея состояла в сравнении слов документа со словами из словаря. Орфографической ошибкой могло оказаться любое отсутствующее в словаре слово. Выглядел сценарий примерно так:

```
cat document |
tr A-Z a-z |          # преобразование к нижнему регистру
tr -d ',.:;()?!' |    # удаление знаков препинания, ...
tr ' ' '\n'           # разбиение слов на строки
sort |                # сортировка слов документа
uniq |                # удаление дубликатов
comm -1 - dict        # отображение строк ввода, которые отсутствуют в словаре
```

Все эти программы уже существовали. Самая необычная из них `comm` позволяла искать строки, которые присутствовали в двух отсортированных входных файлах, или строки, которые присутствовали в одном или другом вводе, но не в обоих сразу. Словарь в папке /usr/dict/web2 содержал слова из второго издания словаря Уэбстера, по одному на строку.

Однажды мне пришлось провести презентацию для Уильяма Колби, который в то время был директором Центрального разведывательного управления (ЦРУ), то есть, несомненно, важным человеком. Его тоже сопровождал Билл Бейкер, который как глава президентского консультативного совета по внешней разведке сам обладал серьезными полномочиями.

Я хотел показать, как Unix упрощает некоторые виды программирования, но сценарий `spell` был достаточно медленным, а я предпочел не затягивать презентацию. Поэтому запустил этот сценарий заранее, записал вывод в файл и написал новый сценарий, который просто «засыпал» на две секунды, а затем выводил результаты, полученные за день до этого:

```
sleep 2
cat previously.computed.output
```

Эта демонстрационная техническая разработка прекрасно себя показала. Если мистер Колби вообще что-то понял, то, наверное, подумал, что проверка правописания была выполнена очень быстро. Но это урок для всех, кто посещает презентации. Не стоит верить всему, что вы там видите!

Отдел связей с общественностью также создавал рекламные фильмы, рассказывающие о чудесах Bell Labs, в том числе и о Unix. Благодаря YouTube я могу посмотреть на старых друзей (и на себя) в то время, когда мы все были моложе, а наши шевелюры гуще.

Была даже небольшая серия печатной рекламы Unix. Насколько я помню, детские кубики на рекламном плакате с рис. 7.4 были моей идеей. Плохо различимый фон — предоставленный мной документ, который был создан в программе Troff.

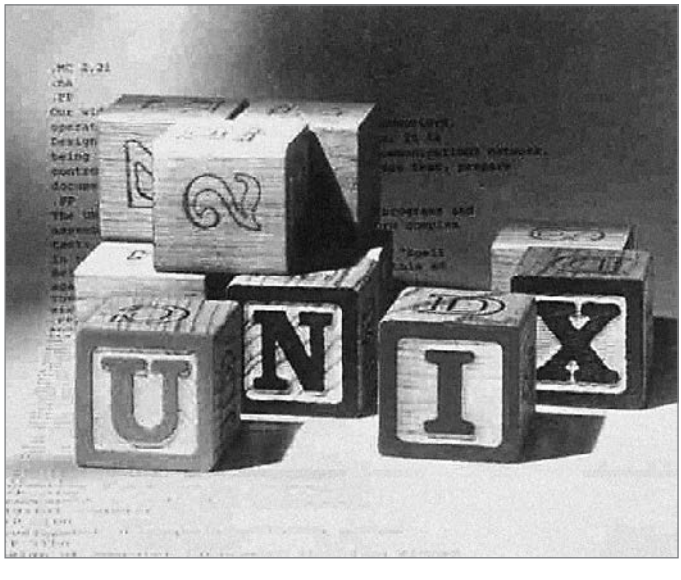


Рис. 7.4. Детские кубики Unix, примерно 1980 год (из архива Bell Labs)

ГЛАВА 8

ПОТОМКИ

> ...из такого простого начала развилось и продолжает развиваться бесконечное число самых прекрасных и самых изумительных форм.
>
> *Чарльз Дарвин, «Происхождение видов», глава 14, 1859 год*

Оперативная система Unix начала свое существование в Исследовательском центре компьютерных наук в 1969 году. Существовали внутренние версии, такие как PWB, которые поддерживали инструменты Programmer's Workbench, но с 1975 года стали появляться и внешние версии, первоначально на базе 6-й, а затем и на базе вышедшей в 1979 году 7-й редакции.

Седьмая редакция стала последним выпуском Research Unix, получившим широкое распространение. После нее были еще три редакции, предназначавшиеся уже для внутреннего использования (они закономерно получили номера 8-я, 9-я и 10-я). Но к концу 1989 года, когда завершилась работа над 10-й редакцией, стало понятно, что основная разработка Unix уже ведется в других местах.

Седьмая редакция породила два ответвления: одно в Беркли на базе работ Билла Джоя и его коллег, а второе в AT&T, где опыт работы с Unix и право собственности на эту операционную систему пытались использовать для построения прибыльного бизнеса. Показанная на рис. 8.1 схема развития приблизительна, там не упоминаются многочисленные версии системы. Реальность была куда более сложной, особенно в аспектах взаимодействия версий.

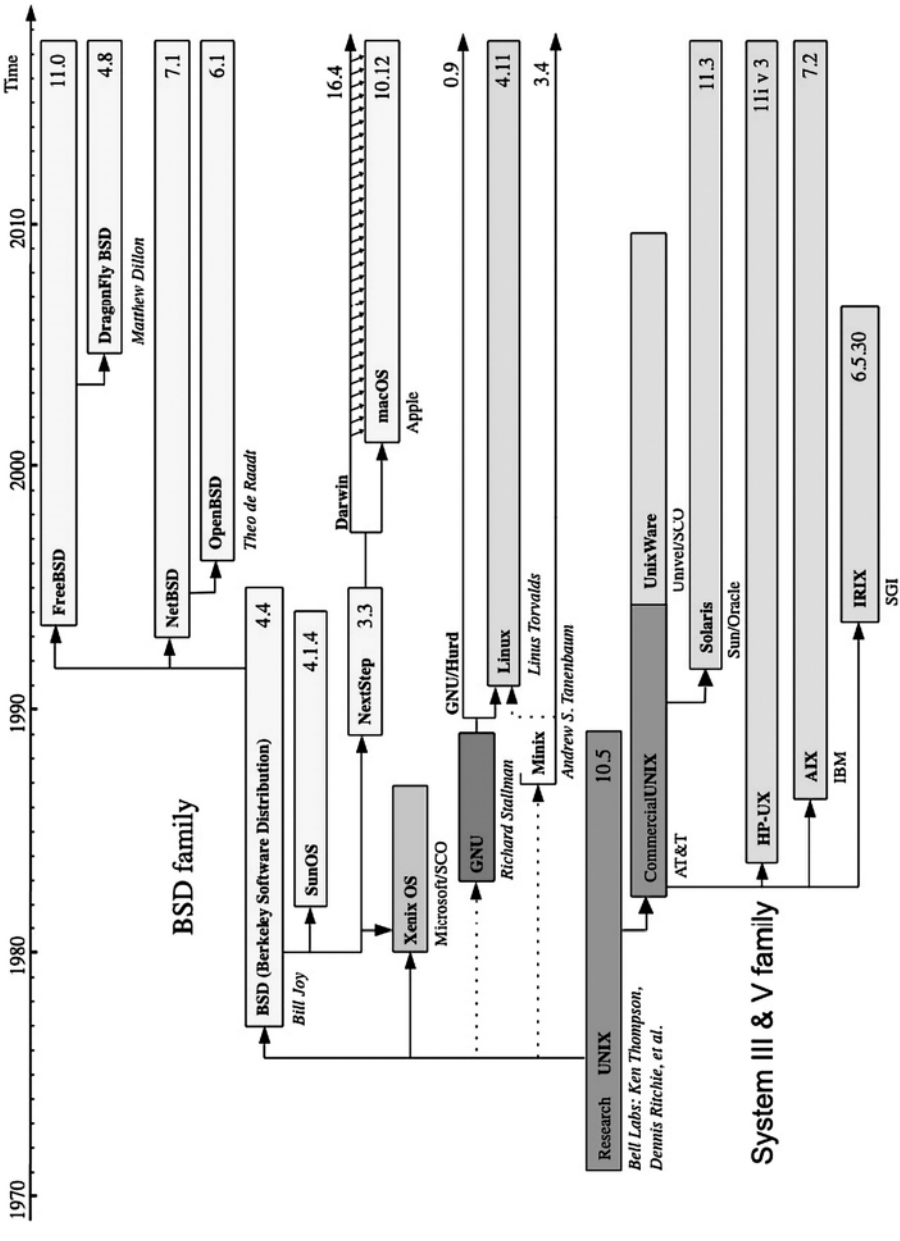


Рис. 8.1. История развития Unix, опубликованная в Википедии

8.1. BERKELEY SOFTWARE DISTRIBUTION

В 1978 году фирма DEC представила новый компьютер VAX-11/780. Это была машина с шириной слова 32 бита, существенно превосходящая PDP-11 по объему памяти и вычислительной мощности, но при этом сохранившая совместимость с PDP-11. После появления VAX-11/780 Джон Райзер и Том Лондон из отдела исследований в Холмделе, штат Нью-Джерси, перенесли на эту машину 7-ю редакцию UNIX. Но эта версия, получившая название 32/V, не задействовала возможностей, которые давала виртуальная память, и, соответственно, не в полной мере пользовалась преимуществами VAX.

Билл Джой и его коллеги из Группы исследований компьютерных систем Калифорнийского университета в Беркли добавили в версию 32/V Райзера и Лондона код, позволяющий пользоваться виртуальной памятью. Новая версия быстро вытеснила 32/V, а компьютер VAX стал основной Unix-машиной для большинства пользователей, которые переросли PDP-11. Лицензиатам Unix систему отправляли как дистрибутив программного обеспечения Беркли (Berkeley Software Distribution, BSD). Потомки BSD по-прежнему активно используются, и даже продолжается разработка таких вариантов, как FreeBSD, OpenBSD и NetBSD. Операционная система NextSTEP, послужившая основой операционной системы Darwin от Apple, компоненты которой используются в macOS, также является производной от BSD.

Один из ранних дистрибутивов Беркли лег в основу операционной системы SunOS, разработанной для своих компьютеров компанией Sun Microsystems, соучредителем которой был Билл Джой. Другие дистрибутивы несколько лет спустя превратились в упомянутые выше варианты BSD. В конечном итоге речь все время шла о повторной реализации, обеспечивающей ту же функциональность с новым кодом. Переписывание избавляло от кода AT&T, соответственно, не нарушались права интеллектуальной собственности AT&T.

Еще один побочный продукт был создан для компании NeXT, основанной Стивом Джобсом в 1985 году. Рабочие станции NeXT обладали функциональностью нового типа и были ранним примером элегантного и изысканного промышленного дизайна, знакомого пользователям Apple. Я присутствовал в аудитории Bell Labs 11 декабря 1990 года, когда Джобс

демонстрировал NeXT. Это была очень хорошая машина. Единственный раз, когда я думал «Я это хочу» о технологическом устройстве. Очевидно, на меня повлияло знаменитое «поле искажения реальности» Джобса. Впрочем, когда три года спустя он проводил в Bell Labs еще одну презентацию, такого эффекта не возникло. Даже не помню, что он показывал.

Хотя коммерческого успеха рабочие станции NeXT не получили, в 1997 году компания была куплена корпорацией Apple, и Джобс туда вернулся, через год став генеральным директором. Наследие операционной системы NextSTEP все еще можно увидеть в именах классов `NSObject` и `NSString` языка Objective-C.

На диаграмме развития можно увидеть еще один малоизвестный факт: в 1980-х годах компания Microsoft распространяла версию Unix, которая называлась Xenix. На рис. 8.2 пример ее рекламы. Интересно, в каком мире мы бы сейчас жили, если бы вместо собственной операционной системы MS-DOS компания Microsoft начала продвигать Xenix и если бы поладить

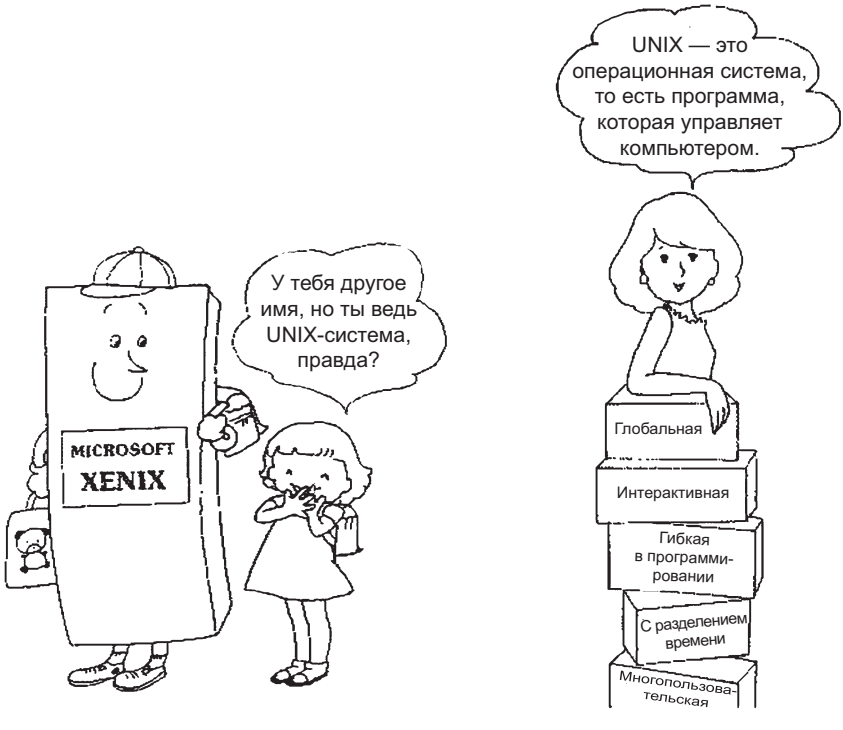


Рис. 8.2. Xenix: версия Unix от Microsoft

с AT&T было проще. По данным сообщества The Unix Heritage Society, впоследствии Xenix купила компания Santa Cruz Operation (SCO), и с середины до конца 1980-х это был наиболее распространенный вариант Unix, если брать количество машин, на которых он был установлен.

8.2. ВОЙНЫ UNIX

В конце 1980-х появилось множество поставщиков Unix, и все они использовали это торговое наименование и закупали ПО, которое первоначально базировалось на 7-й редакции. Однако существовала несовместимость, особенно между System V от AT&T и системами из Беркли. Все стороны были согласны с необходимостью общего стандарта, но никак не могли прийти к соглашению о том, каким он будет.

В 1984 году появился консорциум X/Open с целью создания единой спецификации для производных Unix, чтобы программы можно было без изменений компилировать в любой системе.

Для продвижения своего стандарта, конкурирующего со стандартом группы Open Software Foundation, AT&T с союзниками создали собственную группу Unix International. К чему это привело? Появились два «открытых» стандарта. К мирному соглашению удалось прийти благодаря набору стандартов POSIX (Portable Operating System Interface — переносимый интерфейс операционных систем) для базовых библиотечных функций и стандартам Single Unix Specification от консорциума X/Open, объединившим стандартизированные библиотеки, системные вызовы и множество общих команд (включая командный интерпретатор, язык Awk и редакторы `ed` и `vi`).

В 1992 году USL и AT&T подали в суд на BSDI (Berkeley Software Design, Inc.), обвиняя в разглашении коммерческой тайны и нарушении авторских прав. Впрочем, в Университете Беркли внесли множество изменений в полученный от AT&T код, добавив собственные ценные материалы, в том числе стек протоколов TCP/IP, сделавший интернет доступным.

Процесс удаления и переписывания кода от AT&T продолжался, и к 1991 году появилась версия Unix, которая, по мнению Университета Беркли, уже не содержала никаких собственных материалов AT&T. Но AT&T и USL считали иначе, что и привело к судебному процессу. После долгих интриг

дело было рассмотрено в суде Нью-Джерси, который встал на сторону университета, отчасти на том основании, что AT&T не поместила в распространяемый ими код надлежащие уведомления об авторских правах. Последовали встречные иски.

Сейчас все это кажется чрезвычайно сложным и скучным, но тогда это была огромная проблема, которая привела всех фигурантов к крупным потерям времени и денег. В 1991 году AT&T продала 11 компаниям акции USL, а в 1993 году компанию USL и права на Unix приобрела фирма Novell. Генеральный директор Novell Рей Нурда решил урегулировать все судебные дела, возможно осознав, что вовлеченные в процесс стороны тратят на адвокатов больше, чем могли бы окупить продажи.

Оглядываясь назад, можно сказать, что все эти правовые споры были побочным следствием скороспелого и почти случайного решения AT&T сделать Unix доступной для университетов. Ведь распространившись из университетов, где ее использовали бесплатно, в компании, которые готовы были за нее платить, Unix приобрела коммерческий потенциал. При этом для эффективной охраны авторских прав было уже слишком поздно. Даже после того, как AT&T наложила ограничения на исходный код, в открытом доступе оставался интерфейс системных вызовов, а члены сообщества обладали достаточным опытом, чтобы обычным делом стало создание новых версий, не обремененных лицензиями AT&T. Аналогичная ситуация сложилась с прикладным программным обеспечением, таким как компиляторы, редакторы и все прочие инструменты. Метафорически выражаясь, AT&T пыталась запереть сейф после того, как оттуда унесли все деньги.

8.3. MINIX И LINUX

Из-за попыток AT&T заработать на программном обеспечении лицензия на Unix включала в себя все больше ограничений. В числе прочего она ограничивала использование Unix в университетах, что открывало дорогу BSD. В то же время продолжающиеся войны между AT&T и университетом Беркли побудили других попробовать свои собственные Unix-подобные системы. Независимо создаваемые версии были свободны от коммерческих ограничений, так как использовали только интерфейс системных вызовов, а не чужой код.

В 1987 году в Амстердамском свободном университете Энди Таненбаум создал Minix — аналог Unix, совместимый на уровне системных вызовов, но полностью написанный с нуля с другой структурой ядра.

Оперативная система Minix была сравнительно небольшой, и, чтобы помочь с ее распространением, Энди написал учебник, который в некотором смысле можно было сравнить с появившейся десять лет назад книгой Лайонса. Исходный код Minix был доступен бесплатно. Одна из редакций книги поставлялась с примерно десятком дискет, с которых на IBM PC можно было загрузить работающую систему. У меня все еще хранится первое издание книги Энди, а возможно, даже дискеты с Minix. Сегодня она по-прежнему используется, служа средством для обучения и экспериментирования с ОС.

Результатом ограниченного лицензирования AT&T в сочетании с доступностью Minix в качестве учебного пособия стала еще одна независимая разработка Unix-подобной системы, совместимой на уровне системных вызовов. 25 августа 1991 года 21-летний студент из Финляндии Линус Торвальдс опубликовал в новостной группе Usenet comp.os.minix обращение (рис. 8.3).

```
Здравствуйте, все те, кто использует Minix.
Я делаю (бесплатную) операционную систему (это всего лишь хобби, и она
не будет большой и профессиональной, как gnu) для клонов 386 (486) AT.
Работа над ней началась в апреле и скоро будет закончена. Я бы хотел
получить отзывы о том, что людям нравится/не нравится в Minix, т.к. моя
ОС на нее похожа (такое же устройство файловой системы (из практических
соображений) среди всего прочего).
Я уже перенес bash (1.08) и gcc (1.40), и все, кажется, работает.
Подразумевается, что что-то практическое появится через несколько
месяцев, и я хотел бы узнать, каких особенностей хотелось бы
большинству. Любые предложения принимаются, но я не обещаю, что реализую
их :-) Линус (torv ... @ kruuna.helsinki.fi)
P.S. Да, у нее нет никакого миниксовского кода, и у нее есть
многопоточная фс. Она НЕ переносима (применяет переключение задач 386-го
и т. д.) и, вероятно, никогда не будет поддерживать ничего, кроме
жестких дисков AT, т.к. это все, что у меня есть.
```

Рис. 8.3. Обращение Линуса Торвальдса про Linux, август 1991 года

Линус не ожидал для своей любительской системы замечательного будущего, так же как Кен и Деннис не предвидели успеха Unix. То, что началось с нескольких тысяч строк кода, теперь насчитывает более 20 миллио-

нов строк, а Торвальдс (рис. 8.4) стал основным разработчиком и координатором мирового сообщества разработчиков, которое поддерживает и улучшает этот код. Еще Торвальдс создал Git — наиболее используемую систему контроля версий. Она предназначена для отслеживания изменений кода при совместной разработке программных систем, в том числе и Linux.

Рис. 8.4. Линус Торвальдс в 2014 году (Википедия)

На данный момент Linux — стандартная ОС, способная работать на любом компьютере. Она установлена на миллиардах устройств (например, на всех телефонах Android). Она управляет значительной частью интернет-инфраструктуры, в том числе серверами таких гигантов, как Google, Facebook, Amazon и т. п. Она находится во многих устройствах, реализующих концепцию Интернета вещей (Internet of Things, IoT): под управлением Linux работают мой автомобиль, мой телевизор, ваши Alexa и Kindle и термостаты Nest. Рассмотрев другой конец спектра вычислительных мощностей, вы обнаружите эту ОС на 500 лучших суперкомпьютерах в мире. При этом на рынке ноутбуков и ПК она представлена слабо. Они в большинстве своем работают под управлением Windows и MacOS.

Если говорить о современности, вопрос об авторских правах на программный интерфейс, такой как стандартная библиотека Си или системные вызовы операционной системы, стал центром, по-видимому, бесконеч-

ного судебного процесса между Oracle и Google. В 2010 году корпорация Oracle поглотила компанию Sun Microsystems и получила права на язык программирования Java. Позже в том же году они подали в суд на Google, заявив, что в телефонах Android без разрешения используется защищенный авторским правом Java-интерфейс. Были выдвинуты и другие патентные претензии. Но суд постановил, что патентные претензии недействительны, а Java API (интерфейс прикладного программирования) не может быть защищен авторским правом.

Oracle подала апелляцию, что привело к новому процессу, в котором снова победила Google. Тогда была подана еще одна апелляция, и на этот раз суд вынес решение в пользу Oracle. Компания Google обратилась в Верховный суд в надежде решить там вопрос, можно ли защитить авторским правом API (а не реализации!) с целью предотвратить использование спецификации интерфейса для создания похожих систем.

Должен признаться, что подписал пару экспертных заключений в пользу Google, поскольку не считаю, что API должны защищаться авторским правом. Если бы существовала подобная практика, у нас не появилось бы ни одного аналога Unix, включая Linux, поскольку они основаны на независимой реализации интерфейса системных вызовов Unix.

У нас, вероятно, не было бы и Cygwin — реализации утилит Unix для Windows, которая предоставляет пользователям Windows Unix-подобную среду и интерфейс командной строки. Словом, у нас вряд ли появилось бы много независимых реализаций интерфейсов, если бы претендующие на владение ими компании могли накладывать свои ограничения.

На момент написания этого текста Верховный суд еще не решил, стоит ли рассматривать дело. Но, как только суд примет решение, дело завершится, если только конгресс не внесет изменения в закон. И конечно, никто не знает, что может произойти в других странах.

8.4. PLAN 9

К середине 1980-х работа над Unix в подразделении 1127 начала приостанавливаться. Лежащая в основе большинства внешних версий 7-я редакция появилась в 1979 году. Шесть лет спустя, в 1985 году, была выпущена 8-я ре-

дакция, 9-я — в 1986 году, а 10-я, ставшая последней из версий Research Unix, была завершена в 1989-м и имела хождение исключительно внутри Bell Labs.

Появилось убеждение, что такая зрелая коммерческая система, как Unix, уже не подходит для исследования ОС. Небольшая группа (Кен Томпсон, Роб Пайк, Дэйв Пресотто и Говард Трики) решила поработать над новой операционной системой, которую они назвали «Plan 9 из Bell Labs». В честь научно-фантастического фильма 1959 года «План 9 из открытого космоса», который считался худшей картиной за всю историю кинематографа. Хотя некоторые фанаты фильма считали, что он настолько плох, что в некотором смысле даже хорош.

Операционная система Plan 9 отчасти была попыткой продвинуть вперед хорошие идеи из Unix. Например, в Unix устройства рассматривались как файлы в файловой системе. В Plan 9 в файлы превращалось гораздо большее количество источников данных. В эту группу попадали и процессы, и сетевые подключения, и экраны диспетчера окон, и среда оболочки. Кроме того, новая ОС с самого начала была ориентирована на кроссплатформенность с единым источником, который можно скомпилировать для любой поддерживаемой архитектуры. Еще одной выдающейся особенностью Plan 9 была поддержка распределенных систем. Процессы и файлы в несвязанных системах с разными архитектурами могли работать друг с другом, как если бы находились в одной системе.

В 1992 году новую ОС предоставили в распоряжение университетов, а через несколько лет ее опубликовали для коммерческого использования, но, за исключением небольшого сообщества поклонников, сегодня ее мало кто применяет. Основная причина, вероятно, крылась во все увеличивающейся популярности Unix и Linux, у большинства людей просто не было убедительной причины для перехода на другую систему. Кроме того, Plan 9 могла не прижиться из-за слишком узких рамок. Во многих случаях представленные в ней механизмы были лучше своих эквивалентов из Unix, но попыток обеспечить их совместимость предпринималось немного. Например, изначально в Plan 9 использовали новую библиотеку *bio* вместо стандартной библиотеки ввода-вывода Си. Эта библиотека была более проработанной и регулярной, чем stdio, но без стандартной библиотеки требовалось много усилий, чтобы портировать программы для работы как в Unix, так и в Plan 9. Аналогичным образом появилась новая версия системы сборки Make, которая называлась Mk и была лучше во

многих отношениях, но из-за несовместимости требовалось переписывать существующие make-файлы.

Конечно, механизмы преобразования существовали, и Говард Трики (рис. 8.5) портировал ряд ключевых библиотек, таких как stdio, но даже после этого для многих потенциальных пользователей, включая меня, работа в Plan 9 требовала слишком больших усилий. Таким образом, напрямую использовать большую библиотеку хорошего программного обеспечения из Unix в Plan 9 не получалось. А ее собственные программные инновации было сложно экспортировать в экосистему Unix.

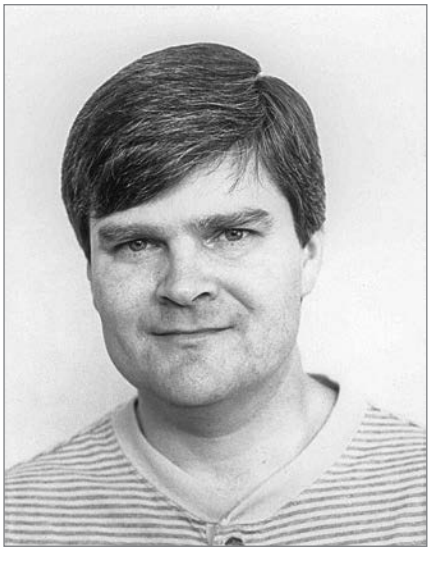


Рис. 8.5. Говард Трики, примерно 1981 год (из архива Джерарда Хольцманна)

Однако благодаря Plan 9 появилось нечто чрезвычайно важное: кодировка UTF-8 в Unicode.

Юникод (Unicod) — это продолжающаяся попытка обеспечить единую стандартную кодировку всех бесчисленных символов, которые человечество когда-либо использовало для написания. Этот символьный набор включает в себя буквенные шрифты, которые используются большинством западных языков, идеографическое письмо, такое как китайские иероглифы, древние шрифты, такие как клинопись, специальные символы и символы всех типов, а также современные нововведения, например эмодзи.

В настоящее время Юникод насчитывает почти 140 000 символов, и их количество медленно, но неуклонно увеличивается.

Изначально Юникод был 16-битным набором символов, достаточно большим, чтобы вместить все алфавитные шрифты и примерно 30 000 символов китайского и японского языков. Но было невозможно заставить всех перейти на 16-битный набор символов, когда большая часть компьютерного текста была представлена в 7- или 8-битном ASCII.

С этой проблемой боролись Кен Томпсон и Роб Пайк, поскольку они решили, что в Plan 9 будет использоваться не ASCII, а Юникод. В сентябре 1992 года они придумали UTF-8 — адаптивную кодировку Юникод переменной длины. Она оказалась эффективной как с точки зрения занимаемой памяти, так и с точки зрения времени обработки. Символы ASCII эта кодировка представляет в виде одного байта, а для большинства других символов использует только два или три, максимум четыре байта. Кодировка компактна, и ASCII является допустимым вариантом UTF-8. Декодирование UTF-8 осуществляется на лету, поскольку среди ее символов нет ни префиксов другого символа, ни частей символа, ни последовательности символов. Почти весь текст в Интернете сегодня закодирован в UTF-8; и она используется всеми и практически повсеместно.

8.5. ПЕРЕСЕЛЕНИЕ НАРОДОВ

В 1996 году снова произошел раскол AT&T, на этот раз добровольный. Компания разделилась на три части, и эта реструктуризация получила название *trivestiture*. Первая часть AT&T занималась междугородной телефонией и связью. Вторая часть Lucent Technologies, ставшая фактически продолжением Western Electric, сосредоточилась на производстве телекоммуникационного оборудования. (Одним из лозунгов компании было «Мы делаем то, что заставляет связь работать».) Третий фигурант деления появился в результате отмены неблагоразумного приобретения компании NCR, которое произошло в 1991 году, когда AT&T пыталась войти в компьютерный бизнес.

В то время сотрудники Bell Labs скептически относились к реструктуризации. Шумиха вокруг нового названия компании и логотипа была встречена с большой долей презрения. На рис. 8.6 показан логотип Lucent, о котором

с большой помпой объявили в 1996 году. Не рискну повторить эпитеты, которые употребляло в его адрес сообщество. На рис. 8.7 вы видите посвященный этому мини-комикс про Дилберта.

Рис. 8.6. Логотип компании Lucent Technologies

Рис. 8.7. Комикс про Дилберта на тему логотипа компании Lucent. ДИЛБЕРТ © 1996 Скот Адамс. Используется с разрешения ANDREWS MCMEEL SYNDICATION. Все права защищены

После этого отдел исследований раскололся по функциональным направлениям: примерно треть персонала ушла в AT&T, сформировав AT&T Labs (сейчас AT&T Shannon Labs), а остальные остались в подразделении Bell Labs, вошедшем в состав Lucent. По большей части люди шли туда, куда им приказывали, но члены подразделения 1127, привыкшие выступать против указов руководства, возмутились насильственным разделением. Мы заняли жесткую позицию, и руководство неохотно позволило каждому самостоятельно сделать выбор. Каждый сам решал, пойти ему в AT&T или остаться с Lucent, но решение принимать нужно было быстро, что стало большим стрессом. В конце концов запланированный раскол все равно

произошел, но у людей хотя бы в краткосрочной перспективе была возможность контролировать ситуацию.

Все стороны в конечном итоге пережили трудные времена. В результате AT&T была приобретена компанией Southwestern Bell (теперь SBC Communications), которая изначально была одной из Baby Bells. SBC забрала себе название AT&T, логотип и даже биржевой код «T», который использовался с 1901 года.

Компания Lucent пережила взлет, а затем падение, поучаствовав по пути в некоторых сомнительных с коммерческой точки зрения мероприятиях. В 2000 году в попытке выжить бизнес в сфере корпоративных коммуникационных услуг вырос в компанию Avaya. А в 2002 году компания, занимающаяся интегральными схемами, была преобразована в компанию Agere. Все это происходило на фоне сокращения финансовой базы, которая могла бы поддержать долгосрочные проекты. В конечном счете Agere была поглощена фирмой LSI Logic, а Avaya, пережив как легкие, так и тяжелые времена и даже банкротство, по-прежнему остается независимой компанией.

В 2006 году Lucent объединилась с французской телекоммуникационной компанией Alcatel, сформировав компанию Alcatel-Lucent, которая в 2016 году перешла во владение Nokia. Компанию Bell Labs охватила волна слияний и поглощений, и большинство сотрудников, связанных с Unix, Центром и подразделением 1127, разошлись в другие места. И даже номер 1127 исчез после реорганизации 2005 года.

На странице www.spinroot.com/gerard/1127_alumni.html находится составленный Джерардом Хольцманном список бывших сотрудников Центра 1127 и их текущего местонахождения. Слишком многие уже умерли, но среди живых больше всего оказались в Google; хотя есть и те, кто работает в других компаниях, преподает или ушел на пенсию. В Bell Labs осталась только малая часть.

ГЛАВА 9

НАСЛЕДИЕ

> Unix стала улучшением не только по сравнению со своими предшественниками, но и по сравнению с большинством следующих систем.
>
> *Дуг Макилрой, замечание к речи Денниса Ритчи на церемонии вручения Премии Японии, май 2011 года, в основу которого легло перефразированное высказывание Тони Хоара про язык Алгол*

Проект Unix оказался чрезвычайно успешным. Под именами Unix, Linux, macOS и прочими операционные системы этого семейства установлены на миллиарды компьютеров, непрерывно обслуживают миллиарды людей и, конечно, помогли заработать миллиарды долларов тем, кто создал на их основе свои коммерческие продукты (хотя к этим людям не относится никто из непосредственных создателей Unix).

Языки и инструменты, изначально разрабатывавшиеся в Bell Labs для Unix, теперь используются повсеместно. Языки программирования Си и C++ до сих пор составляют основу системного программирования, не забыты и более специализированные языки, такие как Awk и AMPL. Базовый инструментарий включает в себя shell, diff, grep, Make и Yacc.

GNU (рекурсивный акроним фразы GNU's not Unix — GNU это не Unix) — это большой набор программного обеспечения, в значительной степени

базирующийся на моделях Unix. Его исходный код бесплатно доступен любому пользователю. Фактически это бесплатный аналог Unix, к которому добавлено многое другое. GNU-реализации команд Unix имеют открытый исходный код, поэтому их можно использовать и расширять при условии, что в случае распространения модифицированных версий исходный код должен быть доступен бесплатно. Огромное количество современных разработок программного обеспечения базируется на открытых исходных кодах, а часто и на реализациях GNU.

Чем объясняется успех Unix? Можно ли извлечь из этой истории какие-то уроки? Думаю, что да, как минимум по двум направлениям — техническому и организационному.

9.1. ТЕХНИЧЕСКИЙ АСПЕКТ

Важные технические концепции, появившиеся благодаря Unix, обсуждались в первых главах. Здесь же я дам их краткое изложение. Конечно, не все, о чем ниже пойдет речь, произошло от Unix: гениальность Кена Томпсона и Денниса Ритчи в числе прочего заключалась в умении выбирать существующие хорошие идеи и в способности видеть общий принцип, упрощающий программные системы. О производительности ПО иногда судят по количеству написанных строк кода; но в мире Unix производительность часто измерялась количеством особых случаев или удаленных строк кода.

Основным упрощением установившейся практики стала *иерархическая файловая система*. С современной точки зрения такая реализация кажется совершенно очевидной. Даже непонятно, как может быть иначе? Вместо файлов разных типов со свойствами, задаваемыми операционной системой, и произвольными ограничениями на глубину вложенности файлов в каталоги файловая система Unix обеспечила простое представление: начиная с корня, каждый каталог содержит либо информацию о файлах, либо каталоги, которые содержат информацию о дальнейших файлах и каталогах. Имена файлов выглядят как путь от корня с каталогами, разделенными косой чертой.

Файлы содержат неинтерпретируемые байты, при этом система не заботится о том, что именно содержится внутри.

Файлы создаются, читаются, записываются и удаляются с помощью простых системных вызовов. Набор битов определяет элементы управления доступом, которых хватает для большинства случаев. К системе можно подключать устройства хранения данных, например съемные диски, после чего остается их смонтировать, и их содержимое станет частью файловой системы.

Естественно, при этом приходится отклоняться от стандартной процедуры. Устройства появляются в файловой системе, что стало большим упрощением, но, так как они не являются файлами в обычном понимании, операции с ними, особенно с терминалами, имеют свои нюансы реализации.

Логическую структуру файловой системы можно реализовать разными способами. Современные системы поддерживают широкий спектр реализаций, причем все они представляют один и тот же интерфейс, но его работу обеспечивают другой код и другие внутренние структуры данных. К компьютеру можно подсоединять различные устройства (жесткий диск, флэш-накопители USB, SD-карты, камеры, телефоны и т. д.), и вы увидите, как для них будут созданы файлы устройств. Удобство Unix заключается в выборе абстракции, достаточно общей, чтобы быть удивительно полезной, но при этом не слишком ресурсоемкой.

Как пользовательские программы, так и сама операционная система написаны на *языке высокого уровня*. Идея была неновой; это пробовали делать при создании Multics и более ранних ОС, но тогда языки высокого уровня были не настолько развитыми, чтобы позволять подобные вещи. Для этих целей Си подходил гораздо больше, чем его предшественники, и именно он дал возможность создать кроссплатформенные реализации операционных систем. Если раньше существовали только проприетарные ОС от производителей оборудования, часто со своими собственными языками, то система Unix стала открытым и широко понятым стандартом, а затем и товаром: система могла использоваться на всех компьютерах с незначительными изменениями. Пользователи получили возможность выбирать ПО без привязки к закрытым решениям производителей.

Командная оболочка, программируемая на уровне пользователя, с инструкциями управления потоком выполнения и простым перенаправлением ввода/вывода позволила использовать программы в качестве строительных блоков. По мере роста возможностей программирования оболочки она превратилась в еще один язык высокого уровня. А так как она не была

фрагментом ОС, ее позволялось улучшать и даже заменять, если появлялась более интересная идея. Переход от оригинального командного интерпретатора Unix через PWB, оболочку Борна и csh Билла Джоя к современным версиям иллюстрирует не только преимущества, но и существенный недостаток: написать новую оболочку слишком легко, что приводит к росту числа несовместимых версий.

Наиболее существенное изобретение Unix — *конвейеры* — элегантный и эффективный способ временного соединения программ друг с другом. Концепция потоковой передачи данных через последовательность этапов обработки легко понимается интуитивно, синтаксис исключительно прост, а механизм конвейера идеально подходит для набора небольших инструментов. Конечно, не все варианты соединения реализуемы, первоначальная концепция Дуга Макилроя — произвольное нелинейное соединение — редко встречается на практике, но линейные конвейеры почти всегда хорошо справляются с поставленными задачами.

Для Unix характерно представление о *программах как об инструментах*. Написание небольших программ, хорошо выполняющих одну задачу, вместо больших программ, которые пытаются выполнять много задач, имеет много преимуществ. Конечно, бывают случаи, когда большие программы имеют смысл, но куда удобнее иметь под рукой набор маленьких инструментов, которые можно комбинировать новыми способами, причем эта операция доступна даже обычным пользователям.

По сути, речь идет о модульности на уровне программ, сходной с модульностью функций внутри программы. Такой подход можно сформулировать как «разделяй и властвуй», поскольку отдельные компоненты не взаимодействуют друг с другом. Он дает пользователю возможность сочетать инструменты по своему вкусу, которую трудно реализовать в большой программе, пытающейся решить слишком много задач в рамках одного пакета.

Обычный текст — стандартный формат данных. Его повсеместное использование стало большим упрощением. Программы читают байты, и если программа предназначена для обработки текста, байты оказываются в стандартном представлении, обычно в виде строк различной длины, завершающихся символом новой строки. Такой подход применим почти всегда, просто в некоторых случаях за него приходится платить небольшим увеличением размера данных и времени обработки. Зато все инструменты

из набора Unix могут как по отдельности, так и в комбинации работать с любыми данными.

Интересно представить, как выглядела бы Unix, если бы на смену перфокартам не пришел телетайп. Фактически перфокарты формируют картину мира, в которой информация разделена на фрагменты по 80 символов и должна помещаться в фиксированные поля внутри этих фрагментов.

Программы, пишущие программы, — удивительно мощный инструмент. Большая часть прогресса в области вычислительной техники достигнута благодаря механизации. Мы смогли заставить компьютер делать работу за нас. Писать программы вручную трудно, поэтому возможность переложить эту задачу на компьютер стала большим шагом вперед. Это сильно упростило процесс, кроме того, код сгенерированных программ практически всегда оказывался корректным.

Первыми в качестве примера на ум приходят компиляторы. На более высоком уровне генерацией кода для создания языков программирования занимаются программы Yacc и Lex. Попадают в эту категорию и инструменты автоматизации и механизации, такие как сценарии оболочки и make-файлы. Они до сих пор широко используются: в частности, дистрибутивы исходного кода языков, таких как Python, и компиляторов, таких как GCC, иногда имеют в своем составе очень большие конфигурационные сценарии и генераторы make-файлов.

Специализированные языки часто называют предметно-ориентированными, или прикладными, языками. Язык позволяет сообщить компьютеру, что мы от него хотим. Большинство программистов пользуется для этого языками общего назначения, такими как Си, но существует и множество более специализированных языков, ориентированных на узкие области.

Хорошим примером в этом отношении является оболочка: она предназначена для запуска программ и прекрасно с этим справляется, но вряд ли кто-то станет писать в ней браузер или видеоигру. Специализация появилась давно; самые ранние языки высокого уровня предназначались для конкретных целей, например Фортран — для научных и инженерных вычислений, а Кобол — для обработки коммерческих данных. Ранние попытки создания языков широкого назначения порой заканчивались неудачей, как в случае с языком PL/I.

В Unix давно используются и языки специального назначения, отличные от оболочки. Это инструменты подготовки документов, с которыми я много работал, калькуляторы, языки проектирования схем, языки сценариев и вездесущие регулярные выражения. Одной из причин появления такого количества языков стали инструменты, позволяющие создавать языки даже неспециалистам. К таким инструментам относятся Yacc и Lex, которые и сами являются специализированными языками.

Язык не обязан быть высокотехнологичным, чтобы приносить пользу. Первую версию команды `at` Стив Джонсон написал за вечер.

В Unix существовал способ запуска задач в нерабочее время, чтобы программы с долгим выполнением не мешали работе остальных (если помните, Unix-машиной пользовалось около десятка сотрудников). Для отложенного запуска задачи нужно было отредактировать системный файл и ввести в таблицу информацию в сложном для понимания формате. Однажды я пытался совершить этот подвиг и бормотал: «Хочу, чтобы эта работа начала выполняться в 2 часа ночи». Внезапно я понял, что эту фразу можно записать с помощью простого синтаксиса: «в 2 часа ночи запустить_эту_команду». За пару часов я написал подходящую версию и на следующее утро объявил об этом через «сообщение дня».

Команда `at` с небольшими изменениями используется и сейчас, хотя прошло уже 40 с лишним лет. Синтаксис любого машинного языка всегда базируется на обычном языке, которым мы пользуемся для общения друг с другом.

Стиль программирования и подход к решению вычислительных задач, получивший название *философии Unix*, был обобщен Дугом Макилроем в 1978 году в предисловии к специальному выпуску издания *The Bell Labs Technical Journal on Unix*:

(i) Пусть каждая программа хорошо делает что-то одно. Для новой задачи напишите новую программу, а не усложняйте старую, добавляя туда новую функциональность.

(ii) Настраивайтесь на то, что выходные данные программы станут входными для другой, пока неизвестной программы. Не засоряйте вывод посторонней информацией. Избегайте ввода исключительно в виде столбцов или в двоичном формате. Не требуйте интерактивного ввода.

(iii) Проектируемое и создаваемое программное обеспечение нужно тестировать на ранней стадии, в идеале в течение первых недель. Без колебаний отбрасывайте плохо сделанные фрагменты и переписывайте их с нуля.

(iv) Облегчайте процесс программирования с помощью специальных инструментов, даже если сначала придется тратить время на их создание, а после завершения работы больше никогда ими не пользоваться.

Это основные принципы программирования, хотя и не всегда соблюдаемые. Например, команда `cat`, о которой я упоминал в главе 3, осуществляла копирование входных файлов или стандартного ввода в стандартный вывод. Сегодня версия команды `cat` для GNU имеет 12 параметров (я не шучу!) для таких задач, как нумерация строк, отображение непечатаемых символов и удаление дубликатов пустых строк. Все эти операции легко выполняются существующими программами, при этом ни одна из них не имеет ничего общего с основной задачей копирования байтов, поэтому они выглядят как непродуктивное усложнение основного инструмента.

Разумеется, философия Unix не решает всех проблем программирования, но это полезное руководство в плане подхода к проектированию и реализации системы.

9.2. ОРГАНИЗАЦИОННЫЕ АСПЕКТЫ

Я считаю, что успех Unix был обусловлен и нетехническими факторами. К ним относятся как управленческая и организационная структура Bell Labs, так и социальная среда в подразделении 1127 и обмен идеями между талантливыми людьми, решающими разные задачи в дружеской обстановке. Эти факторы поддаются оценке сложнее, чем техническая сторона вопроса, поэтому сейчас мой взгляд на ситуацию будет более субъективным. Как и в предыдущем разделе, я перечислю сейчас множество вещей, которые уже упоминались в этой книге.

Решающее значение в таких ситуациях имеет *стабильная среда*: финансирование, ресурсы, миссия, структура, управление, культура — все должно быть последовательным и предсказуемым. Как я упоминал в главе 1, отдел исследований в Bell Labs представлял собой крупную организацию, которая занималась разработками для целой корпорации с долгой историей

и четкой миссией: универсальность обслуживания. Так как Bell Labs планировали постоянно улучшать телефонное обслуживание, разрабатывать идеи, которые сотрудники считали важными, можно было длительное время. Даже если процесс растягивался на годы, никто не заставлял каждые несколько месяцев обосновывать необходимость этой работы. Конечно, контроль был. Если работа над проектом безрезультатно велась несколько лет, исследования настоятельно просили пересмотреть. Иногда кого-то могли освободить от конкретной работы или даже уволить, но за 15 лет на руководящей должности я помню лишь несколько таких случаев.

Финансирование было гарантированным, так что рядовым сотрудникам, занимавшимся исследованиями, не приходилось думать, откуда взять деньги. Даже когда я был начальником отдела, этот вопрос меня не касался. Конечно, где-то сидели люди, в чьи обязанности входила забота об этом, но другие могли целиком посвятить себя исследованиям. При этом не требовалось составлять план работы, писать ежеквартальные отчеты, запрашивать одобрение руководства на каждое свое действие. Хотя, когда я руководил отделом, приходилось составлять полугодовые отчеты о деятельности сотрудников, и я попросил каждого написать несколько строк. Но все это собиралось только для информации, а не для оценки эффективности. Бывали периоды, когда начинали обращать пристальное внимание на поездки, ограничивая количество конференций одной или двумя в год, но по большей части деньги на закупку оборудования и на поездки выделялись без особых вопросов.

Нельзя сбрасывать со счетов и такой важный фактор, как *множество интересных задач*. Как сказал Дик Хэмминг, человек, который не работает над важными проблемами, вряд ли получит значимый результат. В рамках миссии компании AT&T почти любая тема могла оказаться важной и актуальной. Компьютерные науки были новой областью с множеством идей, требующих как теоретического, так и практического изучения. Особенно плодотворным оказывалось объединение теории с практикой. Достаточно вспомнить такие примеры, как инструменты для языков программирования и регулярные выражения.

В AT&T лавинообразно нарастало применение компьютеров, и большая часть этого процесса была связана с Unix, особенно в системах поддержки эксплуатации, как это видно на примере с Programmer's Workbench. Менялся и основной бизнес, так как электромеханические телефонные

коммутаторы уступали место электронным, которые управлялись компьютером. Эта сфера тоже была источником интересных данных и проектов, в которые можно было внести свой вклад. К сожалению, большая часть этой работы выполнялась в крупных филиалах в районе Индиан-Хилл (город Нейпервилл, штат Иллинойс), поэтому сотрудничество требовало поездок в Чикаго. Расстояния и сегодня остаются проблемой. Даже самая лучшая видеосвязь не заменит коллег в соседнем офисе и работающих рядом с тобой экспертов.

Ученые из Bell Labs должны были принимать участие в жизни академического сообщества, так как это был еще один источник интересных проблем и новых знаний. Это позволяло следить за тем, что происходит в других промышленных исследовательских лабораториях, таких как Xerox PARC и IBM Watson. Мы посещали одни конференции, публиковались в одних журналах и часто сотрудничали с коллегами из академических кругов, проводя в вузах творческие отпуска. Например, осенью 1996-го я преподавал в Гарварде при полной поддержке Bell Labs, которые даже платили мне зарплату, так что Гарварду я достался бесплатно. То же самое было в 1999/2000 учебном году в Принстоне.

Мы с коллегами преподавали не только на внутренних курсах, но и в университетах. В случае с близлежащими учебными заведениями, такими как Принстон, Университет Нью-Йорка, Колумбийский университет и Военная академия США, организовать это было очень легко. Впрочем, длительные поездки в более отдаленные уголки устраивались не намного сложнее. Кен Томпсон провел год в Беркли, Роб Пайк — год в Австралии, Дуг Макилрой — год в Оксфорде. Внешние контакты были важны как для набора новых сотрудников, так и для того, чтобы следить за происходящим в отрасли. Не афиширующим себя компаниям труднее привлекать таланты.

Еще одно важное правило: *нанимайте лучших*. За наймом в Bell Labs следили тщательно. В подразделение 1127, как правило, принимали одного или двух человек в год и почти всегда молодых, поэтому решения о найме принимались осторожно, возможно даже чересчур. Эта проблема знакома и факультетам в университетах. Зачастую сложно понять, имеет ли смысл охотиться за звездами в своих областях. Лично я предпочитаю людей, которые действительно хорошо выполняют свою работу, чем бы они ни занимались.

Как бы то ни было, для привлечения хороших специалистов Bell Labs прилагали массу усилий. Агенты по найму из отдела исследований посещали один-два раза в год факультеты компьютерных наук крупных вузов и знакомились с аспирантами. Многообещающих кандидатов на несколько дней приглашали в Bell Labs, где с ними проводили собеседование не только в подразделении 1127, но и в других группах. Помогали в поиске кадров и программы для женщин и меньшинств GRPW и CRFP. Я упоминал о них в главе 1. Оттуда приходили первоклассные кандидаты на постоянную работу, которые уже провели у нас значительное время в период учебы в аспирантуре.

Мы не прибегали к услугам агентств по найму, предпочитая проводить поиск и собеседования своими силами. Когда с преподавателями и студентами общается человек, активно проводящий собственные исследования, он всегда узнает что-то полезное и, кроме того, может оставить положительное впечатление о компании.

Мы старались наладить долгосрочные отношения с университетами. В течение 15 лет я искал новые кадры в Университете Карнеги Меллон в Питтсбурге. Два раза в год я проводил там несколько дней, беседуя с преподавателями факультета компьютерных наук об их исследованиях, а также со студентами, которых могла заинтересовать работа в Bell Labs. Благодаря этому у меня появились хорошие друзья, даже если они и не устраивались к нам на работу. В сфере поиска персонала существовала активная конкуренция, так как свежую кровь активно искали как хорошие университеты, так и ведущие промышленные исследовательские лаборатории. Так что мой список тех, кто ушел в другие места, очень большой. И я был совершенно прав, когда хотел нанять большинство из этих людей; впоследствии они достигли значительных успехов.

Руководители должны *понимать, чем занимаются подчиненные*. Управление исследованиями в Bell Labs вплоть до самого верха осуществлялось людьми, у которых было четкое понимание сути проводящихся работ как в их собственных, так и в других отделах. Подобная осведомленность требовалась не для полемики и попыток навязать свое ви́дение рабочего процесса, а для его подробного объяснения коллегам из других отделов и помощи в установлении связей. По крайней мере, в подразделении 1127 никогда не было борьбы за сферы влияния. У нас сформировалась среда, в которой руководство поддерживало своих людей, сотрудничало и никогда

не составляло конкуренции. Не уверен, что этот опыт можно считать универсальным, но мне кажется, к подобному стоит стремиться.

Несмотря на то что руководство Bell Labs на всех уровнях было технически подкованным, создавалось впечатление, что высшее руководство AT&T не обращает внимания на новую технологию и очень медленно адаптируется к изменениям. Например, в начале 1990-х Сэнди Фрейзер, который тогда возглавлял подразделение 1127, проинформировал руководство AT&T, что совершенствование сетевых технологий означает снижение цен на междугородные звонки с нынешних десяти центов до одного цента в минуту. Но над ним только посмеялись. Сейчас цена стремится к нулю центов в минуту, так что прогноз Сэнди оказался чересчур скромным.

В Bell Labs царил *дух сотрудничества*. Размер и масштабы компании означали присутствие множества экспертов практически в любой технической области, среди них попадались и эксперты мирового уровня. При этом культура общения предполагала готовность помочь. Любой мог зайти в кабинет коллеги и попросить помощи; и чаще всего коллега для этого бросал все свои дела.

У нас была превосходная техническая библиотека, открытая круглые сутки, с подписками на множество журналов и удаленным доступом к другим библиотекам. Это был эквивалент университетской библиотеки с акцентом на науку и технику. Для многих сотрудников подразделения 1127 самыми близкими были эксперты из Центра математических исследований (Mathematics Research Center), или подразделения 1121. Там работали выдающиеся математики: Рон Грэм, Майк Гери, Дэвид Джонсон, Нил Слоун, Питер Шор, Эндрю Одлыжко и др. Кабинет Джона Тьюки — возможно, самого выдающегося статистика своего времени (кстати, именно он придумал слово «бит») — располагался с другой стороны зала, и там можно было встретить самых потрясающих экспертов по практически любым аспектам математики и коммуникаций. Например, мой нынешний коллега из Принстона Боб Тарьян, в 1986 году получивший премию Тьюринга, работал именно в Центре математических исследований.

Эти люди всегда были готовы помочь, и не только по техническим вопросам. Например, Рон Грэм был не только выдающимся математиком, но и опытным жонглером, бывшим президентом Международной ассоциации жонглеров. В его кабинете имелась даже натянутая сетка, чтобы упущенные при жонглировании шары не падали на пол. Рон утверждал, что за 20 минут

может научить жонглировать кого угодно. Боюсь, что оказался бездарным учеником, но через час подробного инструктажа (в его кабинете!) я все-таки смог показать какие-то результаты. У меня все еще хранятся мячи для лакросса, которые он вручил мне для тренировок.

Еще крайне важно *получать удовольствие* от своей работы и своих коллег. Подразделение 1127 почти всегда не только было интересным местом работы, но и позволяло влиться в коллектив замечательных людей. Общение в основном происходило в кафе во время обеденного перерыва, причем дискутировали мы как на технические, так и на социальные темы. Обитатели комнаты Unix обычно обедали в 13:00, в то время как большая группа предпочитала делать это с 11:00. Обсуждалось все, от больших и малых технических идей до политики. Беседы часто продолжались и во время прогулок вокруг зданий Bell Labs.

Члены Центра то и дело устраивали розыгрыши и слишком сильно увлекались противодействием бюрократии, которая неизбежна в любой крупной компании. Про бейджи я уже говорил. Дополнительные возможности открывали различные формы и процедуры, которым мы должны были следовать. Например, сотрудники службы безопасности покупали машины, нарушающие те или иные правила. Однажды весной Майк Леск нашел незаполненную квитанцию на штраф и положил ее на лобовое стекло автомобиля коллеги, написав: «до 1 апреля не было снято крепление для перевозки лыж». Коллега (я не буду называть здесь его имени) потратил несколько часов, чтобы снять крепление.

Безусловно, самую сложную шутку сыграли с Арно Пензиасом, которого пригласили «протестировать» новое программное обеспечение для распознавания голоса. Шутку готовила команда из минимум десятка человек с Робом Пайком и Деннисом Ритчи во главе, а помогали нам профессиональные иллюзионисты Пенн и Теллер. Подробности Деннис описал на странице www.bell-labs.com/usr/dmr/www/labscam.html, а тут можно посмотреть видео: www.youtube.com/watch?v=if9YpJZacGI. Я фигурирую в титрах как осветитель (gaffer), что вполне соответствует действительности: мне пришлось использовать много тканевой клейкой ленты (gaffer tape).

Бесплатной еды в лабораториях не было (этот бонус появился куда позже, хотя тогда мы бы его очень оценили), но каким-то образом у нас был бесплатный кофе. За него платило руководство.

В комнате Unix оставляли различные предметы общего пользования. Кто-то нашел у себя запас из десяти килограммов высококачественных шоколадных плиток и принес, чтобы люди их съели. Хотя бывали и нестандартные случаи:

> Кто-то принес пакет с надписью на китайском языке. Все мы пробовали эти непонятные штуки и плевались. Затем мы заметили, что они исчезают: оказалось, что этот человек их ест. В нижней части пакета сотрудник, который знал китайский язык, прочитал инструкцию, гласившую, что перед едой нужно в течение часа замачивать их в кипятке.

Никаких мероприятий по формированию навыков коллективной работы и командных игр, которые сегодня получили такое распространение, у нас не проводилось. Большинство из нас считало их искусственными и бессмысленными и смотрело на них как на бесполезную трату времени.

Для создания и поддержания организации, члены которой любят и уважают друг друга, требуется много усилий. Подобное нельзя создать ни руководящими указаниями, ни с помощью внешних консультантов. Такие вещи органично возникают от удовольствия работать вместе, иногда вместе шутить и ценить хорошо сделанную работу коллег.

9.3. ПРИЗНАНИЕ

Работа Кена Томпсона и Денниса Ритчи, основных создателей Unix, получила широкое признание. В 1983 году комитет по отбору кандидатов на премию Тьюринга высказался так:

> Успех системы проистекает из тщательного выбора нескольких ключевых идей и их элегантной реализации. Пример системы UNIX привел поколение разработчиков программного обеспечения к переосмыслению основ программирования. Основной принцип системы UNIX заключен в ее подходе, который позволяет программистам опираться на работу других.

В 1999 году Кен и Деннис получили Национальную медаль США в области технологий. На фото 9.1 вы видите президента Билла Клинтона и одну из немногих ситуаций, когда Кен или Деннис надевали костюм и галстук. По

меркам того времени обстановка в Bell Labs была крайне неформальной. Как писал в своей онлайн-биографии Деннис, «у Кена много связей в верхах. Я один из немногих, кроме Бонни Т., кто неоднократно видел его в костюме (и даже с черным галстуком)». Лично я никогда не видел Кена при параде.

Рис. 9.1. Кен, Деннис, Билл, Национальная медаль в области технологий, 1999 год

Другие награды включают членство в Национальной инженерной академии США и Премию Японии в области информации и коммуникаций в 2011 году, для которой было написано следующее:

По сравнению с ранее существовавшими операционными системами, новая, усовершенствованная ОС проще, быстрее и имеет удобную иерархическую файловую систему. Одновременно с Unix велась разработка языка программирования Си, который до сих пор широко применяется для написания ОС и значительно улучшил читабельность и переносимость исходного кода Unix. В результате Unix получила широкое применение: встроенные системы, персональные компьютеры и суперкомпьютеры.

Еще Unix стала главной движущей силой развития интернета. Калифорнийский университет в Беркли разработал Berkeley Software Distribution (BSD) — расширен-

ную версию Unix, реализованную с использованием набора протоколов интернета TCP/IP. Разработка базировалась на шестой редакции Unix, которую Bell Labs вместе с исходным кодом распространяли среди университетов и исследовательских институтов в 1975 году, что породило культуру «открытого исходного кода». Операционная система BSD Unix помогла в реализации интернета.

Другие формы признания были менее формальными. Unix и язык Си стали частью популярной культуры; например, Си упоминали в популярной телевикторине *Jeopardy!*:

```
От dmr@cs.bell-labs.com вторник 7 января 02:25:44 2003
Тема: на случай, если вы этого не видели
Вечером в пятницу в игре “Jeopardy!” в категории “Letter Perfect” (ответ на любой
вопрос должен состоять из одной буквы) самый сложный вопрос на $2,000 был:
ОН РАЗРАБОТАН В НАЧАЛЕ 1970-Х И ЯВЛЯЕТСЯ ОСНОВНЫМ ЯЗЫКОМ ПРОГРАММИРОВАНИЯ
ОПЕРАЦИОННОЙ СИСТЕМЫ UNIX.
```

Ну и в качестве кульминации для особых фанатов напомню знаменитую сцену из фильма «Парк Юрского периода» 1993 года, где 13-летняя Лекс Мерфи (ее играет Ариана Ричардс) говорит: «Это система Unix! Я ее знаю». Она перемещается по файловой системе, находит элементы управления дверями, запирает их и тем самым спасает всех от велоцирапторов (рис. 9.2).

Рис. 9.2. Unix в «Парке Юрского периода»

Получили профессиональное признание и другие члены Центра 1127, отчасти благодаря богатым возможностям, которые предоставила Unix. Восемь бывших сотрудников подразделения 1127 являются членами в Национальной инженерной академии США.

9.4. МОЖЕТ ЛИ ИСТОРИЯ ПОВТОРИТЬСЯ?

Возможна ли новая Unix? Может ли операционная система появиться из ниоткуда и за несколько десятилетий захватить мир? Когда я рассказываю о Unix, мне часто задают этот вопрос. И я отвечаю: нет, по крайней мере не сейчас. Новой революции не будет. Скорее всего, операционные системы продолжат развиваться, сохраняя значительную часть ДНК Unix.

Но в других областях вычислительной техники аналогичный успех вполне возможен. До сих пор встречаются изобретательные и одаренные люди, попадаются хорошие руководители, кроме того, сейчас аппаратное обеспечение очень дешевое, а отличное ПО зачастую можно получить вообще бесплатно. С другой стороны, благоприятствующая подобному обстановка стала скорее редкостью. Промышленные исследования сильно сокращаются, и сейчас они куда более краткосрочные, чем 50 лет назад. А научные исследования еще и ограничены в средствах.

Однако я настроен оптимистично, потому что считаю, что великие идеи исходят от людей.

Например, в первые дни существования Unix ею занимались всего два человека. Ядро написал Кен Томпсон — безусловно, лучший программист из когда-либо встречавшихся мне, человек, не имеющий себе равных по нестандартности мышления. Большой вклад внес и Деннис Ритчи, а придуманный им язык программирования Си, который послужил средством эволюции Unix, до сих пор остается *общепринятым языком* для информационно-вычислительных систем.

Полезно посмотреть историю популярных сейчас языков. Вы увидите, как часто на начальном этапе ими занимались один или два человека. Это касается практически всех основных языков программирования: Java (Джеймс Гослинг), C++ (Бьёрн Страуструп), Perl (Ларри Уолл), Python (Гвидо ван Россум) и JavaScript (Брендан Эйх). Можно гарантированно предсказать,

что будут появляться новые языки, чтобы сделать программирование проще и безопаснее. Также можно с уверенностью утверждать, что универсального языка под все задачи не появится никогда. Невозможно написать язык, хорошо подходящий для всего.

В основе Google, Facebook, Amazon, Twitter, Uber и многих других компаний, которые из стартапов превратились в многомиллиардные предприятия, лежат яркие идеи одного или двух человек. Таких компаний будет все больше, хотя новые перспективные стартапы могут оперативно скупаться существующими крупными компаниями. Яркие идеи все равно получают развитие, их авторы — хорошую награду, но мальков быстро съедает большая рыба.

Грамотное управление — еще одна составляющая успеха. Здесь выделяется Дуг Макилрой как уникальный лидер с выдающимися интеллектуальными способностями, с несравненными техническими оценками. Его стиль руководства основывался на том, что он становился одним из первых пользователей того, что было разработано его коллегами. Сама Unix, языки Си и C++ и множество инструментов Unix получили возможность усовершенствования благодаря хорошему вкусу и резкой критике Дуга. Аналогичная ситуация сложилась со всей документацией по Unix, от руководств пользователя до нескольких десятков книг.

Я могу лично это засвидетельствовать. В 1968 году Дуг был рецензентом моей диссертации, безжалостно комментировал все мои технические статьи и книги и даже сейчас, более пятидесяти лет спустя, помогает советами.

Руководство Bell Labs было технически компетентным, особенно в подразделении 1127, что позволяло ценить хорошую работу и придерживаться политики невмешательства. Не было продавливания конкретных проектов или подходов. За 30 лет работы в Bell Labs мне никогда не говорили, чем мне заниматься. Брюс Ханней, вице-президент отдела исследований, второе лицо после Билла Бейкера, написал в 1981 году в издании *Engineering and Science in the Bell System*:

Для исследователя первостепенное значение имеет свобода выбора, потому что он изучает неизвестное. Когда нет карты, невозможно понять, какой курс следует выбрать. Каждое открытие влияет на будущее направление исследований, предсказать или запланировать открытие невозможно. Руководи-

тели отдела исследований в Bell Labs предоставили персоналу максимально возможную свободу, что согласуется с целями учреждения. Люди в отдел исследований отбирались за их творческие способности, и им предлагается использовать эти способности в полной мере.

Одним из лучших примеров почти абсолютной свободы, которую я когда-либо видел, стала работа Кена Томпсона и Джо Кондона над шахматным компьютером Belle. Однажды президент Bell Labs Билл Бейкер привел в комнату Unix очередного важного посетителя, и Кен постарался произвести впечатление, продемонстрировав Belle. Посетитель спросил, какое отношение работа над компьютерными шахматами имеет к телефонной связи. Бейкер ответил, что Belle — эксперимент с компьютерами специального назначения, который позволил разработать новые инструменты для проектирования интегральных микросхем и реализации схем и сделал Bell Labs известными в другой области. Так что Кену не приходилось подтверждать правомерность своей работы.

Большой секрет успешных исследований заключается в том, чтобы нанять хороших специалистов, убедиться, что у них есть интересные темы для разработки, понаблюдать за ними, а потом отойти. Идеала в этом, конечно, достичь не удалось, но исследования в Bell Labs шли хорошо.

Разумеется, вычислительная техника не может существовать в технологическом вакууме. Изобретение транзистора, а затем интегральных схем привело к тому, что за 50 лет вычислительное оборудование становилось все меньше, быстрее и дешевле, причем этот процесс шел с экспоненциальной скоростью. По мере улучшения аппаратного обеспечения упростились разработка и использование ПО, а мы стали лучше понимать, как его создавать. Как и многие другие системы, Unix развивалась на волне совершенствования технологий.

Как я уже говорил в предисловии, рождение Unix, которое изменило мир компьютеров, вероятно, было сингулярностью, возникшей благодаря уникальной комбинации обстоятельств. Сомневаюсь, что в области ОС мы снова увидим что-то подобное, но наверняка будут и другие случаи, когда горстка талантливых людей с хорошими идеями и благоприятной средой изменит мир своими изобретениями.

Для меня Bell Labs и подразделение 1127 стали изумительным опытом: временем и местом с бесконечными возможностями и группой первоклассных

коллег, которые умели по максимуму использовать эти возможности. Немногим повезло иметь такой опыт. Разумеется, все это наша общая заслуга, и я благодарен друзьям и коллегам, которые принимали в этом участие.

Мы хотели сохранить не просто хорошую среду для программирования, но и систему, вокруг которой могло сформироваться братство. По своему опыту мы знали, что суть общего доступа к вычислительным ресурсам [...] не в том, чтобы вводить программы не с перфокарт, а нажимая клавиши, а в том, чтобы стимулировать тесное общение.

Деннис Ритчи, статья *The Evolution of the Unix Time-sharing System*, октябрь 1984 года

ИСТОЧНИКИ

> Имена Ритчи и Томпсона можно с уверенностью связать со всеми случаями, кроме тех, когда в явном виде указывается другое авторство.
>
> Более подробный рассказ невозможно было бы уместить в отчет, потребовалось бы написание целого тома, причем желательно не участником событий, а более беспристрастным ученым.
>
> *Дуг Макилрой, A Research Unix Reader: Annotated Excerpts from the Programmer's Manual, 1971–1986, 1986 год*

Большая часть истории операционной системы Unix выложена в интернете (хотя и не всегда в доступной для поиска форме), благодаря усилиям как любителей, так и профессиональных историков, например членов группы *The Unix Heritage Society* и работников Музея компьютерной истории. Дополнительный материал доступен в многочисленных видеоинтервью и в виде устных историй, как современных, публикуемых отделом связей с общественностью AT&T, так и ретроспективных. Ниже приведен список англоязычных ресурсов, с которых можно начать дальнейшее знакомство с историей Unix. В интернете выложено множество документов.

Сотрудниками Bells Labs в 1970-х и 1980-х годах написаны семь томов *A History of Science and Engineering in the Bell System*. Это почти 5000 страниц текста. Один из томов посвящен вычислительным системам, которые появились относительно поздно.

Работавший в 1960-х и в начале 1970-х годов в Центре речевых и акустических исследований (Speech and Acoustics Research Center) А. Майкл Нолл написал воспоминания о годах, проведенных в Bell Labs. Эти воспоминания наряду с материалами, полученными при редактировании бумаг Билла Бейкера, можно найти на сайте noll.uscannenberg.org, как и множество другой полезной исторической информации. Там вы найдете основные факты, связанные с Bell Labs, и познакомитесь с областью исследований речи и акустики. Воспоминания Майкла о товарищеском духе и открытости среди сотрудников в целом совпадают с моими, хотя, по его версии, все стало исчезать гораздо раньше. Возможно, потому что мы работали в разных (хотя и смежных) областях.

Аналогичный архив информации об операционной системе Multics поддерживает Том Ван Флек на странице multicians.org.

Июль 1978 года ознаменовался специальным выпуском журнала *Bell System Technical Journal*. Он был посвящен Unix и содержал ряд фундаментальных статей, в том числе обновленную версию статьи *Unix Implementation*, которую Кен написал для журнала CACM, статью *The UNIX Time-sharing System — A Retrospective* Денниса, статью Стива Борна об оболочке и посвященные PWB статьи Теда Долотты, Дика Хайта и Джона Мэши.

В 1984 году в специальный выпуск журнала *Bell Labs Technical Journal* вошли статьи *Evolution of Unix* Денниса Ритчи, *Data Abstraction in C* Бьёрна Страуструпа и другие интересные материалы.

Снабженные аннотациями выдержки из руководства пользователя *A Research Unix Reader* Дуга Макилроя дают особенно хорошее представление об истории вопроса; их можно найти на сайте genius.cat-v.org/doug-mcilroy.

Группа The Unix Heritage Society, которой руководит Уоррен Туми, с помощью многочисленных добровольцев сохранила ранние версии кода и документации Unix. Например, по адресу www.tuhs.org/Archive/Distributions/Research/Dennis_v1/ вы найдете код первой редакции, предоставленный Деннисом Ритчи.

Покойный профессор истории науки Майкл Махони из Принстона за лето и осень 1989 года взял интервью у десятка сотрудников подразделения 1127 для написания истории Unix. Необработанные стенограммы и отредактированные интервью хранятся на историческом факультете Принстона и до-

ступны по адресу www.princeton.edu/~hos/Mahoney/computing.html. Майкл был не только первоклассным историком, но и программистом, и хорошо понимал, о чем говорили интервьюируемые, поэтому собранные им материалы содержат множество технических деталей.

Филлис Фокс, пионер численных вычислений и одна из первых женщин среди технического персонала Bell Labs в 2005 году, написала материал для некоммерческого научного общества Society for Industrial and Applied Mathematics (SIAM). Этот материал, доступный по адресу history.siam.org/oralhistories/fox.htm, в числе прочего содержит подробное описание разработанной в Лабораториях библиотеки для языка Фортран PORT.

Беседа с Кеном Томпсоном, состоявшаяся в мае 2019 года на фестивале старинных компьютеров, выложена на YouTube по адресу www.youtube.com/watch?v=EY6q5dv_B-o.

Для бесплатного скачивания доступны две книги о ранней истории Unix. Это *Life with Unix*, написанная Доном Либсом и Сэнди Весслер в 1989 году, и *A Quarter Century of Unix* Питера Салуса, вышедшая в 1994 году.

Сохранилась домашняя страница Денниса Ритчи на сайте Bell Labs http://www.bell-labs.com/usr/dmr/www/. Там есть ссылки как на большинство статей Денниса, так и на другие исторические материалы.

Один из основных разработчиков BSD Кирк Маккьюзик написал подробную историю этой операционной системы, которая доступна по адресу https://www.oreilly.com/openbook/opensources/book/. Дополнительную информацию и немного другой взгляд на происходящее дают на странице doc.cat-v.org/unix/unix-before-berkeley/ Ян Дарвин и Джефф Колье.

Брайан Керниган

Время UNIX. A History and a Memoir

Перевела с английского И. Рузмайкина

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*К. Тульцева*
Литературный редактор	*А. Руденко*
Обложка	*В. Мостипан*
Корректоры	*Н. Сулейманова, Г. Шкатова*
Верстка	*Е. Неволайнен*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.
Дата изготовления: 09.2020.
Наименование: книжная продукция.
Срок годности: не ограничен.
Налоговая льгота — общероссийский классификатор продукции ОК 034-2014,
58.11.12 — Книги печатные профессиональные, технические и научные.
Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.
Подписано в печать 24.08.20. Формат 70x100/16. Бумага офсетная. Усл. п. л. 18,060. Тираж 700. Заказ

КНИГА-ПОЧТОЙ

ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **books@piter.com**
- по телефону: **(812) 703-73-74**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

- Наложенным платежом с оплатой при получении в ближайшем почтовом отделении.
- С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
- Электронными деньгами. Мы принимаем к оплате Яндекс.Деньги, Webmoney и Qiwi-кошелек.
- В любом банке, распечатав квитанцию, которая формируется автоматически после совершения вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ:

- Посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку ваших покупок максимально быстро. Дату отправления вашей покупки и дату доставки вам сообщат по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте www.piter.com).
- Можно оформить доставку заказа через почтоматы,(адреса почтоматов можно узнать на нашем сайте www.piter.com).

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

БЕСПЛАТНАЯ ДОСТАВКА:

- курьером по Москве и Санкт-Петербургу при заказе на сумму **от 2000 руб.**
- почтой России при предварительной оплате заказа на сумму **от 2000 руб.**