Содержание

От авторов перевода

Признаться, мы не сразу решились взяться за перевод книги профессоров Никлауса Вирта и Юрга Гуткнехта, полагая, что ее содержание, уходящее корнями в далекие 80 – 90-е годы прошлого века, вряд ли сможет заинтересовать современного читателя. Однако по мере знакомства с ней наши сомнения стали быстро рассеиваться, пока не стало понятно, что в книге излагается оригинальный и – не побоимся этого слова – свежий, из первых рук взгляд на, казалось бы, давно известную тему и, главное, богатейший практический материал. Решающий удар по нашим сомнениям был нанесен, когда при чтении книги мы натолкнулись на приведенные ее авторами слова одного из основоположников дисциплины программирования К.Хоара: «В нашей отрасли считается, что инженеры должны постоянно создавать новые артефакты без возможности *изучения предыдущих работ, знание которых как раз и должно доказывать их ценность для отрасли*». Для нас стало очевидным, что медлить с переводом этой книги просто преступно: ее идеи, ее глубокое и насыщенное содержание обязательно должны выйти за пределы узкой «касты жрецов» и стать достоянием широкого круга профессионалов и не только их. Мы с азартом взялись за перевод и закончили его так быстро, что он оказался некоторой неожиданностью для авторов, которые намеревались не спеша «омолодить» для последующих изданий свою уникальную книгу.

А книга, вне всяких сомнений, во многом уникальна. Устав распутывать бесчисленные узелки хитросплетений самых «совершенных» на то время операционных систем, ее авторы предприняли отчаянную, дерзкую и, как они сами выразились, «безумную» попытку *спроектировать и реализовать* ***с нуля*** собственную *операционную систему* для персональной рабочей станции. И эта попытка была ими блестяще осуществлена (причем всего за три года!) в проекте Оберон, результаты которого во всех подробностях представлены в этой книге-отчете. Результаты проекта действительно значительные.

Прежде всего, «побочным», но отнюдь не второстепенным, продуктом проекта стал новый язык программирования Оберон – достойный продолжатель своих знаменитых предков Паскаля и Модулы-2. Унаследовав от них все самое лучшее и избавившись от лишнего, он стал стройнее и лаконичнее: его полный синтаксис занимает не более двух страниц. В то же время он стал мощнее за счет встроенного в него механизма расширения типов, позволяющего программировать практически любые расширяемые системы (каковой является и сама система Оберон),

придерживаясь при этом объектно-ориентированной парадигмы. Его строжайшая типизация (в том числе указателей) позволяет теперь проектировать, описывать и разрабатывать исключительно надежные программные системы, не опасаясь за то, что какой-нибудь «атипичный» указатель заведет вычислительный процесс, например, в запретную зону памяти, как это бывало раньше. О «внешних влияниях» на язык Оберон Н.Вирт с легкой иронией говорил: «Просто невозможно поблагодарить всех тех, кто так или иначе подпитывал своими идеями то, что теперь называется Оберон. Большинство идей пришло от использования и изучения таких существующих языков, как Модула-2, Ада, Smalltalk и Cedar, которые часто предостерегали нас от того, как не надо делать».

Не последнюю роль в поддержке доброго имени языка Оберон играют его компилятор, загрузчик и сборщик мусора. Компактный и быстрый компилятор создает достаточно плотный код. «Умный» динамический загрузчик не позволяет размножаться в оперативной памяти многочисленным копиям одних и тех же исполняемых модулей, и, кроме того, внимательно следит за версиями исполняемых модулей, предоставляя тем самым возможность их совместной разработки. А сборщик мусора, как неотъемлемый компонент управления динамической памятью, обеспечивает оперативное автоматическое освобождение основной памяти от «хлама».

Благодаря этим и другим своим широким и мощным возможностям, язык Оберон становится в руках разработчика средством спецификации, надежной реализации и детального документирования больших программных проектов, как и сам проект Оберон, то есть может служить в качестве языка системного программирования высокого уровня. Вместе с тем, благодаря своей семантической ясности, синтаксической простоте и лаконичности на грани аскетизма, он может стать, пожалуй, наилучшим среди прочих языком для обучения дисциплине программирования, то есть служить в качестве языка систематического программирования.

Несмотря на то, что представленная в книге полная реализация системы Оберон написана на языке Оберон, его строгого определения вы в ней не найдете. Отчасти и по этой причине в нашем предисловии мы просто не имели права обойти его вниманием и не посвятить ему хотя бы несколько похвальных строк. Однако читатель даже с небольшим опытом программирования при внимательном чтении найдет в книге все необходимые сведения о языке (особенно в главе 12, посвященной его компилятору) и легко разберется в текстах программ. Тем же, кто хочет почувствовать «аромат» и «вкус» языка, можно порекомендовать, например, книгу Н. Вирта «Алгоритмы и структуры данных: Новая версия для Оберона» и прилагаемый к ней CD с популярным открытым вариантом системы Оберон – системой программирования BlackBox Component Builder с базовым языком Компонентный Паскаль – усовершенствованным диалектом языка Оберон.

Но главный «персонаж» книги – конечно же, сама система Оберон, представляющая собой очищенную от «мути» и «вредных примесей», компактную, полнофункциональную, многозадачную операционную систему для персональной рабочей станции, которая, будучи дополненной (расширенной) ее авторами сетевыми возможностями, может послужить основой для мультисерверной станции.

К счастью, авторы фактически освободили нас от труда вводить читателя в курс дела, поскольку сами поставили перед собой и с блеском решили непростую задачу по описанию операционной системы с нуля во всех ее деталях, как говорят, от «а» до «я». Мы посчитали, что давать здесь свои пространные пояснения к представляемой в книге системе (даже такой сложной, как операционная) было бы проявлением неуважения к высочайшему профессионализму, с которым авторы задумали и осуществили свой проект. Поэтому львиная доля наших усилий была направлена именно на достойный этой книги перевод, для того чтобы здесь ограничиться только несколькими общими комментариями.

Главный недостаток широко распространенных коммерческих операционных систем – в том, что все они наследуют и, главное, преумножают пороки своих предков. Их развитие (если не сказать, разбухание) – это постоянное наращивание новых возможностей вокруг некогда отлаженного и проверенного ядра, зачастую путем объединения нескольких отдельных специализированных систем в одну, более мощную и универсальную. При этом в погоне за скорым результатом у разработчиков коммерческих систем практически нет времени, чтобы оглянуться назад и разобраться во всем том, что было сделано раньше, а главное понять, насколько грамотно и хорошо все это было сделано. (Впрочем, перед ними такая задача и не ставится.) В итоге, за гигантские размеры «операционок», за неэффективность их работы, за расточительство памяти расплачиваются, увы, многочисленные их пользователи, вынужденные приобретать под них компьютеры с еще более мощными процессорами и еще бо́льшей оперативной памятью.

Таким образом, практически все коммерческие операционные системы подтверждают закон Райзера: несмотря на свой стремительный прогресс «железо» ускоряется медленнее, чем замедляется «софт». Проектом Оберон авторы попытались опровергнуть этот закон, и эта попытка им, похоже, удалась. Вот их вывод: «система Оберон потребовала очень малых усилий для создания коммерческих операционных систем широкого назначения и скромных запросов к скорости и объемам памяти компьютера при той же мощности и гибкости для пользователя, хотя и без некоторых излишеств».

Высокая «себестоимость» системы Оберон проявилась уже хотя бы в том, что многие оригинальные идеи этого, в сущности, открытого проекта стремительно разлетелись по многим более поздним коммерческим системам. По этому поводу в одной из своих лекций Н.Вирт, с присущим ему чувством юмора, говорил, что разработчики языка Java за несколько лет до его создания «изучали исходные коды Оберона, в частности, коды его сборщиков мусора. Потом испортили Оберон синтаксисом Си и назвали получившееся словом «Java». Несмотря на это, хорошо структурированная, модульная и потому компактная и надежная система Оберон вместе со своим базовым языком Оберон заняла достойное место в ряду других систем и по сей день продолжает успешно развиваться и совершенствоваться, не теряя при этом своего ясного, открытого и выразительного «лица».

Первой идеи проекта Оберон начала активно развивать и продвигать основанная в 1992 году коллегами Н.Вирта по ETH компания Oberon Microsystems, одним из членов совета директоров которой стал сам Н.Вирт. К настоящему вре-

мени в мире, в том числе в России, существует и используется несколько успешных операционных систем и удобных сред разработки и поддержки программ на языке Оберон или его диалектах, «отцом» которых стал описанный в книге оригинальный проект. Так, сама компания Oberon Microsystems разработала компактную и «ресурсосберегающую» среду визуального программирования BlackBox Component Builder для диалекта языка Оберон, названного Компонентным Паскалем. Для разработки и исполнения программ на языках уже семейства Оберон (Оберон-2, Оберон 07 и др.) были разработаны и другие операционные системы и среды, «производные» от первоначальной системы Оберон, – ETH Oberon, WinOberon, Bluebottle, A2 System и другие. Многие из них успешно используются в самых разных областях техники – от бортовых систем управления полетом беспилотных летательных аппаратов и гражданских авиалайнеров до систем управления сложной медицинской техникой. Появились также намерения воплотить систему Оберон в «железе» – в виде микросхемы на одном чипе. Все, кто заинтересуется этими и еще более «юными» потомками проекта Оберон, смогут легко отыскать их по ключевым словам на страницах всемирной «паутины».

Система Оберон может быть смонтирована с помощью прилагаемого к ней набора инструментов на абсолютно «голом» компьютере как самостоятельная операционная система или «посажена» на «родную» операционную систему компьютера как удобная интегрированная среда разработки и исполнения программ на собственном базовом языке семейства Оберон.

Несмотря на исчерпывающую полноту изложения материала, присущую техническим отчетам, авторы временами переходят на конспективный стиль в расчете на то, что читатель знаком с основными (фундаментальными) понятиями информатики.

Надеемся, что все интересующиеся этой темой читатели прочтут книгу с таким же увлечением, как и ее переводчики. Надеемся также, что она окажется нужной и полезной как ветеранам отрасли, так и начинающим. Для первых она может стать глотком свежего воздуха, придать второе дыхание, а для вторых – допингом для новых «безумных» идей и дерзновенных свершений.

И, конечно же, вне всяких сомнений, богатейший материал книги (в полном объеме или частично) можно и должно рекомендовать для изучения любых дисциплин системного программирования, в которых рассматривается широкий круг вопросов – от проектирования до реализации операционных систем.

В заключение, несколько слов о переводе.

Операционным системам присуща широта охвата самых разнообразных тем и понятий – от низкоуровневых, конкретных (сигналов на линиях передачи данных, протоколов, растровых операций на экране дисплея) до высокоуровневых, абстрактных (абстрактных типов данных, изощренных алгоритмов разметки и сборки «мусора»). Поэтому попытка соединить в одном, довольно небольшом по объему, тексте «стихи и прозу, лед и пламень» уже сама по себе представляет собой многотрудную задачу не только для его авторов, но и для его переводчиков. Но если авторов к преодолению многочисленных трудностей в их подвижнической работе подстегивала их «безумная» идея, то нас в более чем скромной работе –

«безумный» интерес к их идее, на волне которого наши трудности в итоге оказались почти смешными.

Так, например, весь текст книги пронизывает красной нитью фундаментальное понятие системы Оберон «viewer». Похоже, авторы просто в воду глядели, когда утверждали в главе 4: «несмотря на то, что все, казалось бы, пришли к согласию в значении этого слова, на деле любые два системных разработчика расходятся в его трактовке». Именно это и случилось с нами, когда мы искали подходящий перевод этому многозначному слову. В итоге, отказавшись от труднопроизносимого «просмотрщика», от двусложного «средства просмотра», от уже занятого другими системами «окна» и от многих-многих других, мы остановились на простом и несколько наивном, но коротком слове «окошко».

Другой пример – также многозначное слово «master». В контексте разных глав его значение, в идеале, должно быть разным. Однако после длительных согласований мы не нашли ничего лучшего, как «присвоить» ему единое, не зависящее от контекста, значение другого (тоже латинского) слова «клиент», то есть тот, кто чего-то просит или требует и оттого, видимо, всегда прав.

Заканчивая, мы хотим принести свои извинения профессиональным инженерам-схемотехникам за возможные неточности перевода специальных терминов в главе 9, касающейся тех аппаратных средств и внешних устройств компьютеров, технических описаний которых у нас не было (да и не могло быть) под рукой.

Москва, декабрь 2011 г. *Е. Борисов, Л. Чернышов*

Предисловие

Эта книга представляет результаты проекта Оберон, а именно полную программную среду для современной рабочей станции. Проект был предпринят авторами в 1986–1989 годах, и его главная цель состояла в том, чтобы спроектировать и реализовать всю систему с нуля (на пустом месте) и построить ее таким образом, чтобы она могла быть описана, объяснена и понята как единое целое. Чтобы вскрыть все аспекты, проблемы, проектные решения и детали, авторы не только придумали, но, более того, запрограммировали всю описанную в книге систему.

Хотя существует множество книг, объясняющих принципы и структуру операционных систем, ощущается нехватка описаний систем, которые на самом деле реализованы и используются. Мы хотели не только дать совет, как может создаваться система, но и показать, как она была создана. В связи с этим ключевую роль в книге играют тексты программ, так как только они содержат окончательные объяснения. Поскольку в этом случае выбору удобного формализма придавалась особая важность, мы разрабатывали язык Оберон не только как эффективный инструмент реализации, но и как средство публикации алгоритмов, подобно созданному три десятилетия назад Алголу 60. Благодаря своей структуре язык Оберон также хорошо подходит и для отображения глобальной модульной структуры программируемых систем.

Несмотря на небольшое количество человеко-лет, затраченных на реализацию системы Оберон, и несмотря на ее компактность, позволяющую уместить ее описание в одной книге, это не академическая игрушка, а скорее универсальная система для рабочей станции, которая нашла множество довольных и даже восторженных пользователей в академической среде и в промышленности. Описанное здесь ядро системы, состоящее из управления памятью, файлами, отображением, текстом, окошками (viewers), из загрузчика программ и драйверов устройств, черпает свою главную мощь из удачно выбранного, гибкого набора основных средств и, что более важно, из их эффективной расширяемости во многих направлениях и для многих приложений. Расширяемость чрезвычайно усиливается языком Оберон, с одной стороны, и эффективностью основного ядра, с другой. Она коренится в применении объектно-ориентированной парадигмы, которая используется везде, где расширяемость оказывается выгодной.

В дополнение к основной системе мы описываем во всех деталях компилятор языка Оберон и графическую систему, которые могут рассматриваться как приложения. Первый показывает, как разработать компактный компилятор, чтобы достичь и быстрой компиляции, и эффективного, компактного кода. Последняя приводится в качестве примера расширяемой разработки, основанной на объект-

но-ориентированных методах, и показывает возможность ее тесной интеграции с существующей текстовой системой. Еще одно дополнение к основной системе – сетевой модуль, позволяющий множеству рабочих станций взаимодействовать между собой. Мы также показываем, насколько система Оберон, будучи дополненной распределением файлов, печатью и электронной почтой, удобна в качестве основы для мультисерверной станции.

Компактность и регулярная структура, а также должное внимание эффективной реализации важных деталей оказываются ключом к экономичной инженерии программного обеспечения. Системой Оберон мы хотим опровергнуть закон Райзера, который был подтвержден фактически всеми недавними выпусками операционных систем: несмотря на большие скачки вперед, аппаратные средства ускоряются медленнее, чем замедляется программное обеспечение. Система Оберон потребовала очень малых усилий для создания коммерческих операционных систем широкого назначения и скромных запросов к скорости и объемам памяти компьютера при той же мощности и гибкости для пользователя, хотя и без некоторых излишеств. Приглашаем читателя узнать, как это стало возможно.

Но еще важнее то, что мы надеялись представить заслуживающее внимания исследование значительной части программирования вообще для пользы всех тех, кто стремится учиться на опыте других.

Мы хотим поблагодарить многих людей за их предложения, советы и поддержку. В частности, наших коллег Х. Мессенбека и Б. Сэндерса и наших товарищей из Института вычислительных систем (Institut für Computersysteme) за прочтение всей или части рукописи этой книги. Мы благодарны М. Брэндису, Р. Крелье, А. Дистели, М. Францу и Дж. Темплу за их работы по успешному переносу системы Оберон на различные коммерческие компьютеры, делающие таким образом предлагаемое исследование более интересным для многих читателей. Мы также благодарны нашей школе ETH за вклад в обеспечение такой обстановки и поддержки, которые позволили нам заняться этим проектом и довести его до конца.

Цюрих, февраль 1992 г.
Н. В. и Ю. Г.

1. ИСТОРИЯ И МОТИВАЦИЯ

Можно ли заставить себя сосредоточиться на работе в теплый день под роскошным солнцем и синим небом? Этим риторическим вопросом я не раз задавался, проводя отпуск в Калифорнии в 1985 году. Любой чувствовал бы себя обязанным вернуться домой полным удовольствий от жизни в деревне, путешествий или занятий любимым спортом в такие солнечные дни. Но здесь каждый день был таким, и поддаться подобному соблазну значило бы положить конец всей работе. И не я ли сам выбрал это место в мире за его привлекательный, приятный климат?

К счастью, моя работа была так же соблазнительна, облегчая тем самым мою участь. Я имел честь восседать перед самой передовой и мощной, где бы то ни было, рабочей станцией и постигать секреты, возможно, новейшей причуды в ремесле скоростной разработки, гоняя по экрану цветные прямоугольники. Все это должно было происходить по строгим правилам, налагаемым физическими законами и новейшими технологиями. К счастью, передовой компьютер тут же выражал недовольство, если эти правила нарушались. Это был контролер правил, который подобно старшему брату предупреждал вас о шагах, ведущих к беде. Он делал то, что было бы невозможно сделать самому, отслеживая тысячи связей между тысячами размещенных на экране прямоугольников. Это называлось машинным проектированием (computer-aided design), или проектированием с *помощью* компьютера. «Помощь» – это скорее эвфемизм, хотя компьютер не жаловался на принижение его роли в этом процессе.

В то время как мои глаза были прикованы к многоцветью экрана и я со всей очевидностью оказался перед лицом своей крайней неосведомленности, в открытую всегда дверь шагнул мой коллега. Оказалось, он тоже проводил свободное от домашних дел время в этой лаборатории, но лицо его выражало не столько счастье, сколько огорчение. Плитка шоколада в его руке была для него тем же, чем для других чашка кофе или флейта, давая временное расслабление и отвлечение. То был не первый случай, когда он появлялся в таком настроении, и я без слов угадывал его причину. И это могло бы повторяться еще не раз.

Его дни не были заполнены забавами с прямоугольниками; у него была цель – разработать компилятор для этого самого передового компьютера. Поэтому он вынужден был знать основную программную систему намного ближе, если не глубже. В его положении столь частые неудачи должны были быть поняты, поскольку он программировал, а я лишь использовал систему в приложениях, то есть был конечным пользователем! Эти неудачи нужно было понять не для того, чтобы их

исправить, а чтобы найти способ избежать их. Но как достичь необходимого понимания? В этот момент я понял, что пока далек от этого вопроса; я ограничил знакомство с этой новой системой до необходимого минимума, который был достаточен для моей задачи.

Вскоре стало ясно, что разобраться в системе было почти невозможно. Ее размеры были просто устрашающими, а документация – довольно скудной. Ответы на неотложные вопросы лучше всего было получать из расспросов разработчиков системы, которые всегда были рядом. При этом мы сделали потрясающее открытие: часто мы не могли понять их язык. Их объяснения были полны жаргона и ссылок на другие части системы, которые оставались для нас такими же загадочными.

Таким образом, наши перерывы в работе, вызванные расстройствами от конструкции компилятора и процессора, стали посвящаться попыткам уяснить суть, принципы новых аспектов системы. Чем она отличается от обычных операционных систем? Какие из ее концепций существенны, а какие можно улучшить, упростить или даже отбросить? И где именно они заложены? Можно ли извлечь суть системы и очистить ее, как в химическом процессе?

Во время последующих обсуждений потихоньку возникала идея предпринять наш собственный проект. И вдруг она обрела реальность. «Безумие, это невозможно» было моей первой реакцией. Полный объем работы казался непреодолимым. В конце концов, мы оба должны были выполнять дома и свои преподавательские обязанности. Но мысль уже засела в нас и продолжала занимать наши умы.

Чуть позже домашние обстоятельства сложились так, что мне нужно было принять важный курс системного программирования. Поскольку по неписаным правилам он должен был иметь дело прежде всего с принципами построения операционных систем, я колебался. Мои сомнения были легко объяснимы: ведь я никогда не разрабатывал ни систему в целом, ни даже ее часть. А как можно преподавать инженерную дисциплину без личного опыта!

Неужели невозможно? А разве мы не проектировали компиляторы, операционные системы и редакторы документов малыми командами? И разве я многократно не сталкивался с тем, когда не вполне подходящая и неудачная программа переписывалась с нуля как часть исходного кода оригинального проекта? Наш мозговой штурм продолжался со многими перерывами более чем несколько недель, и сквозь туман стали медленно проступать определенные контуры структуры системы. Некоторое время спустя безумное решение было принято: мы начнем с нуля проект операционной системы для нашей рабочей станции (которая, как оказалось, обладала гораздо меньшей мощностью, чем та, на которой я гонял по экрану прямоугольники).

Основная цель – накопить собственный опыт и достичь полного понимания каждой детали – в сущности, определила наши трудовые ресурсы: два частично занятых программиста. Предварительно мы установили себе срок завершения работ в три года. Как выяснилось позже, оценка была верной: программирование началось в начале 1986 года, а первая версия системы была выпущена к концу 1988 года.

Хотя поиск подходящего названия для проекта – это обычно вопрос второстепенный и зачастую дело случая или прихоть разработчиков, было бы уместно рас-

сказать, как в поле нашего внимания попал Оберон. Случилось так, что мы начали наш проект, когда космический зонд Voyager заполнял передовицы газет сериями своих захватывающих снимков планеты Уран и ее спутников, наибольший из которых назывался Оберон. С этого момента я рассматривал проект Voyager как исключительно хорошо спланированное и успешное предприятие, и в качестве скромной дани ему я выбрал имя его последнего объекта исследования. На самом деле есть совсем немного инженерных проектов, результаты которых выходят за пределы ожиданий и ожидаемой их жизни; по большей части они терпят неудачу гораздо раньше, особенно в области программного обеспечения. И последнее, но немаловажное: напомним, что Оберон известен как король эльфов.

Сознательное ограничение трудовых ресурсов заставило нас принять единственное, но здравое решение: сосредоточиться на основных функциях и пренебречь красотами, которые лишь угождают укоренившимся традициям и преходящим вкусам. Конечно, в первую очередь должно быть определено и оформлено основное ядро. Хотя основание уже было заложено. Наш руководящий принцип стал еще более важным, когда мы решили, что результат должен будет использоваться в качестве обучающего материала. Я помнил призыв К. Хоора (C. A. R. Hoare) о том, что книги по операционным системам должны быть конкретными, а не представлять собой описание полусырых, абстрактных принципов. В начале 70-х он сетовал: в нашей отрасли считается, что инженеры должны постоянно создавать новые артефакты без возможности изучения предыдущих работ, которое должно доказывать их ценность для отрасли. Как же он был прав, даже сегодня!

Возникшая цель – опубликовать результат во всех его деталях – заставила по-новому отнестись к выбору языка программирования: он стал решающим. Язык Модула-2, которым мы планировали воспользоваться, оказался не очень подходящим. Во-первых, потому что ему недоставало средств адекватного выражения расширяемости, а мы провозгласили расширяемость одним из основных принципов новой системы. В «адекватность» мы включаем машинную независимость. Наши программы не должны зависеть от особенностей машины и низкоуровневых средств программирования, за исключением, быть может, интерфейсов устройств, где такой зависимости не избежать.

Поэтому язык Модула-2 был расширен свойством, которое теперь известно как *расширение типа* (*type extension*). Мы выяснили также, что язык Модула-2 содержал несколько средств, которые не нужны и на самом деле не способствуют его выразительной силе, но в то же время усложняют компилятор. А компилятор должен был быть не только реализован, но и описан, изучаем и понятен. Это привело к решению начать с пересмотра языка реализации проекта и применить к нему тот же принцип: сосредоточиться на главном, избавляясь от лишнего. Новому языку, который все еще имел много общего с Модула-2, дали то же имя, что и всей системе, – Оберон [1, 2]. В отличие от его предка, он лаконичнее и, главное, является значительным шагом к выражению программ на высоком уровне абстракции независимо от особенностей машины.

Мы начали разработку системы в конце 1985 года, а программирование – в начале 1986 года на нашей рабочей станции Lilith и ее языке Модула-2. Сначала был

создан кросс-компилятор, а за ним – модули внутреннего ядра вместе с необходимыми средствами тестирования и загрузки. Одновременно шла разработка системы отображения и текстовой системы без возможности их тестирования, конечно. Мы поняли, насколько отсутствие отладчика и, более того, компилятора может способствовать тщательному программированию. (*Это действительно так, в чем убедился один из нас, когда примерно в то же самое время и примерно в тех же условиях писал компиляторы языка C. – Прим. перев.*)

Затем последовал перевод компилятора на язык Оберон. Это было сделано стремительно, потому что оригинал был написан с намерением последующего перевода. После его проверки на целевом компьютере Ceres вместе со средствами редактирования текста пуповина Lilith могла быть отрезана. Система Оберон, по крайней мере, ее черновая версия, стала реальной. Это случилось примерно в середине 1987 года; после этого было опубликовано ее описание [3].

Завершение системы заняло еще год, ушедший на объединение рабочих станций в сеть для передачи файлов [4], на средства централизованной печати и на инструменты поддержки. Наша цель – завершить систему в три года – была достигнута. В середине 1988 года система была представлена более широкому сообществу пользователей, и можно было начать работу над приложениями. Была разработана почтовая служба, добавлена графическая система и продолжены различные работы по общим системам подготовки документов. Средство отображения было расширено так, чтобы работать с любым экраном, включая цветной. Одновременно на основе опыта использования системы совершенствовались отдельные ее части. С 1989 года в наших вводных курсах программирования язык Модула-2 был заменен языком Оберон.

Здесь, видимо, уместно сказать об используемом оборудовании. Рабочая станция Ceres тоже была разработана в Институте вычислительных систем ETH, что обеспечило идеальную почву для внедрения системы Оберон на «голой» машине. Это дало безграничную возможность проектировать без оглядок на установленные ограничения и избегать компромиссов, вызванных несовместимой средой.

Ceres-1 была создана на микропроцессоре NS-32032 фирмы National Semiconductor, который в 1985 году был первым коммерчески доступным процессором с 32-разрядной шиной. Его удобная система команд оказалась особенно привлекательной для разработчика компилятора. Компьютер был оснащен 2 Мб оперативной памяти, жестким диском на 40 Мб, дисководом, дисплеем с разрешением 1024×800 пикселей и, конечно, клавиатурой и мышью. Этих ресурсов было более чем достаточно для системы Оберон.

Ceres-2 была создана в 1988 году заменой процессора его более быстрой версией NS-32532, который увеличил ее вычислительную мощность по сравнению с предшественницей почти в 5 раз. Память была увеличена до 4–8 Мб, а диск до 80 Мб. Для установки программного обеспечения было переделано «всего» несколько модулей: ядро (из-за иной структуры страницы) и драйверы устройств (из-за иных адресов устройств).

В 1990 году была разработана недорогая версия Ceres-3, и 100 таких компьютеров были созданы и установлены в лабораториях. Этот одноплатный компью-

тер построен на процессоре NS-32GX32 без устройства виртуальной адресации с 4–8 Мб оперативной памяти. Его отличительная черта заключается в том, что файловая система реализована в одной (защищенной) половине оперативной памяти вместо диска, резко повышая скорость ее работы. Ceres-3 свободны от механических устройств (даже вентилятора) и потому совершенно бесшумны. Они используются прежде всего в лабораториях для студентов. Польза центрального сервера для распределения системных файлов очевидна.

Успех системы, ее гибкость дали в 1989 году начало проекту по переносу системы на многие коммерчески доступные рабочие станции. От плана устанавливать систему на «голых», как Ceres, машинах быстро отказались: никто бы и не стал делать таких попыток, если для экспериментов с Обероном нужно было покупать другой компьютер или хотя бы менять ROM. Минусы надстройки над существующей системой нужно было принять; они представляют собой некое редко используемое программное обеспечение, занимающее часть памяти, иногда значительную. На момент написания этой книги существовали реализации на Apple Macintosh II, Sun Microsystem Sparc Station, DEC Station 3100 и 5000 и IBM RS/6000. Каждая из этих реализаций занимала примерно половину человеко-года. Решение о надстройке над существующей системой имеет неоценимое преимущество: приложения, созданные в основной системе, доступны из Оберона. Все они отвечают своим описаниям из руководства пользователя [Reiser, 1991], имеют тот же пользовательский интерфейс, и каждая программа, работающая на одном из этих компьютеров, может без изменений выполняться на любом другом. Очевидно, это – важное преимущество, которое может появиться только при программировании на более высоком уровне абстракции, как в языке Оберон.

Литература

1. N. Wirth. The programming language Oberon. *Software – Practice and Experience 18*, 7, (July 1988) 671–690.
2. M. Reiser and N. Wirth. *Programming in Oberon – Steps beyond Pascal and Modula*. Addison-Wesley, 1992.
3. N. Wirth and J. Gutknecht. The Oberon System. *Software – Practice and Experience, 19*, 9 (Sept. 1989), 857–893.
4. N. Wirth. Ceres-Net: A low-cost computer network. *Software – Practice and Experience, 20,* 1 (Jan. 1990), 13–24.
5. M. Reiser. *The Oberon System – User Guide and Programmer's Manual*. Addison-Wesley, 1991.

2. ОСНОВНЫЕ ПОНЯТИЯ И СТРУКТУРА СИСТЕМЫ

2.1. Введение

Для оправдания значительных усилий по проектированию и разработке операционной системы с нуля в ее фундаментальных понятиях должно быть много новизны. Мы начнем эту главу с обсуждения основных понятий, лежащих в основе системы Оберон, и главных проектных решений. Далее на этом базисе будет представлена структура системы, причем в самом общем виде, а именно: состав и взаимозависимость ее крупных блоков, или модулей. Глава заканчивается кратким обзором последующих разделов книги. Это должно помочь читателю понять роль, место и значение частей системы, представленных отдельными главами.

Основная цель операционной системы – представить компьютер пользователю и программисту на определенном уровне абстракции. Например, память представляется в виде требуемых участков или переменных указанного типа данных, диск – в виде цепочек символов (или байтов), именуемых файлами, дисплей – в виде прямоугольных областей, именуемых окошками (viewers), клавиатура – как входной поток символов, а мышь – как пара координат и набор состояний ее кнопок. Каждая абстракция характеризуется определенными свойствами и управляется набором операций. Задача системы – выполнять эти операции и управлять ими в пределах доступных ресурсов основного компьютера. Это обычно называется управлением ресурсами.

Каждая абстракция скрывает в себе подробности, причем именно те, от которых она абстрагируется. Сокрытие может происходить на разных уровнях. Например, компьютер в зависимости от своего режима работы (обычный/привилегированный) может запретить доступ к определенным участкам памяти или определенным устройствам. То же самое можно сделать с помощью средств сокрытия языка программирования с его правилами видимости. Последнее, конечно, намного гибче и мощнее, и первое на самом деле играет почти незаметную роль в нашей системе. Сокрытие важно, потому что оно гарантирует сохранность определенных свойств абстракции, именуемых инвариантами. По сути, абстракция – ключ к модульности, а без модульности исчезает всякая надежда на возможность добиться надежности и правильности. Ясно, что система Оберон проектировалась с целью установления модульной структуры на базе целенаправленных абстракций. Наличие соответствующего языка программирования – необходимая предпосылка для этого, а важность его выбора невозможно переоценить.

2.2. Понятия

2.2.1. Окошки

В то время как абстракции отдельных переменных, представляющих участки оперативной памяти, и файлов, представляющих участки дисковой памяти, являются хорошо устоявшимися понятиями и имеют значение в любой вычислительной системе, абстракции устройств ввода-вывода приобрели свою значимость с повышением взаимодействия (интерактивности) между пользователем и компьютером. Высокая интерактивность требует широкой полосы пропускания, а единственный широкополосный канал пропускания у человека – это глаз. Следовательно, устройство визуализации вывода компьютера должно было хорошо подходить для человеческого глаза. Это произошло в середине 1970-х с появлением дисплея с высокой разрешающей способностью, что, в свою очередь, стало возможным благодаря более быстрым и более дешевым электронным компонентам памяти. Дисплей с высоким разрешением знаменовал собой один из немногих, очень существенных прорывов в истории компьютерных технологий. Типичная полоса пропускания современного дисплея – порядка 100 МГц. Именно дисплей с высоким разрешением сделал визуальный вывод предметом абстракции и управления ресурсами. В системе Оберон дисплей разделен на окошки, именуемые также окнами, или, точнее, на кадры – прямоугольные области экрана. Обычно окошко состоит из двух кадров – полоски заголовка с именем объекта окошка и меню команд и основного кадра с неким текстом, графикой, изображением или другим объектом. Само окошко – это кадр; кадры могут быть вложенными, в принципе, до любой глубины.

Система обеспечивает процедуры генерации кадра (окошка), его перемещения и закрытия. Она размещает новое окошко в определенном месте, а по запросу дает подсказку, куда его лучше всего поместить. Она следит за множеством открытых окошек. Это так называемое управление окошками, в отличие от управления их содержимым.

Однако высокая интерактивность требует не только широкополосного пропускания визуального вывода, но и гибкости ввода. Конечно, нет никакой необходимости в одинаково широкой полосе пропускания, поскольку клавиатура, ограниченная скоростью набора около 100 Гц, для этого не годится. Прорывом на этом фронте стала так называемая мышь, указывающее устройство, которое появилось примерно в то же время, что и дисплей с высоким разрешением.

Это было не только удачным совпадением. Мышь возможна только на дисплее с высоким разрешением и с надлежащим программным обеспечением. Концептуально сама по себе она – очень простое устройство, подающее сигналы при движении по столу. Эти сигналы заставляют компьютер обновлять положение указателя (курсора) на дисплее. Так как обратную связь поддерживает глаз человека, от мыши не требуется высокой точности. Например, когда пользователь хочет указать на некоторый объект на экране, скажем, символ, он перемещает мышь, пока курсор не достигнет этого объекта. Она заметно контрастирует с цифровым датчиком, который, как предполагается, должен предоставить точные координаты. Система Оберон во многом полагается на возможности мыши.

Возможно, самая хитроумная идея заключалась в том, чтобы снабдить мышь кнопками. Имея возможность одной рукой и перемещать курсор, и подавать команды, пользователь может сразу увидеть результат их выполнения. Позиционная зависимость реализована в программном обеспечении делегированием обработки сигнала процедуре (так называемому обработчику или интерпретатору), которая локализуется в том окошке, куда хотя бы на миг попадает курсор. Таким образом, соответствующим программным обеспечением может быть достигнута удивительная гибкость активации команд. В связи с этим появились различные технологии, например всплывающие меню, ниспадающие меню и т. д., которые возможны даже при наличии всего одной кнопки. Для многих приложений мышь с несколькими кнопками еще лучше, а система Оберон в основном предполагает наличие трех кнопок. Назначение кнопкам разных функций, конечно, может легко привести к путанице, когда каждое приложение подразумевает разное назначение кнопок. Но этого легко избежать, если придерживаться определенных «глобальных» соглашений. В системе Оберон левая кнопка прежде всего используется для пометки позиции (устанавливая символ вставки), средняя – для выполнения общих команд (см. ниже), а правая – для выделения отображаемых объектов.

Сейчас стало модным использовать наложение окон, отображающих стопку документов на столе. Мы нашли эту метафору не очень убедительной. Перед тем как выполнить некую операцию над содержимым частично скрытого окна, оно обычно поднимается наверх и делается полностью видимым. В итоге незначительное достоинство несопоставимо со значительными усилиями, необходимыми для реализации такой схемы. Это хороший пример того, когда выгода от усложнения несоразмерна его цене. Поэтому мы выбрали решение, которое гораздо проще реализовать и которое, тем не менее, не имеет реальных недостатков по сравнению с перекрывающимися окнами, – мозаичные окошки, как показано на рис. 2.1.

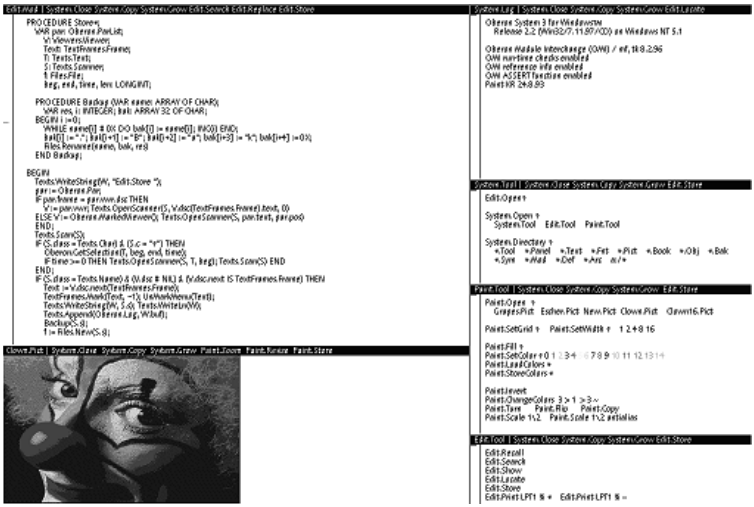

Рис. 2.1. Дисплей Оберона с мозаичными окошками

2.2.2. Команды

Позиционно-зависимые команды с фиксированным (для каждого типа окошка) смыслом должны быть дополнены общими командами. Обычно такие команды подаются с клавиатуры набором в специальном окне команд имени программы, которая должна быть выполнена. В этом отношении система Оберон предлагает новое и гораздо более гибкое решение, которое приводится в следующих параграфах.

Прежде всего отметим, что программа – в самом общем смысле текст, скомпилированный в единицу исполнения, – это обычно довольно большой блок действий, чтобы быть только командой. Сравните ее, например, со вставкой части текста по команде мыши. В Обероне понятие единицы действия отделено от понятия единицы трансляции. Первая – это команда, представленная (экспортируемой) процедурой, а последняя – это модуль. Следовательно, модуль может определять и, как правило, определяет несколько, а то и множество команд. Такая (общая) команда может быть вызвана в любое время указанием на ее имя в любом тексте, видимом в любом окошке на дисплее, и нажатием средней кнопки мыши. Имя команды имеет вид *M.P*, где *P* – идентификатор процедуры, а *M* – идентификатор модуля, где объявлена *P*. Как следствие, любой щелчок на команде может вызвать загрузку одного или нескольких модулей, если *M* еще не загружен в основную память. Следующий вызов *M.P* происходит мгновенно, так как *M* уже загружен. Более того, модули никогда не удаляются (автоматически), потому что следующая команда вполне может обратиться к тому же самому модулю.

Каждая команда имеет целью изменение состояния некоторых операндов. Обычно они следуют за идентификатором команды, и Оберон выполняет это соглашение. Строго говоря, команды обозначаются как процедуры без параметров, но система дает им возможность идентифицировать самих себя в тексте и, следовательно, прочитать и проинтерпретировать последующий текст, то есть их фактические параметры. Однако и чтение, и интерпретация должны программироваться явно.

Текст параметров должен ссылаться на объекты, которые уже существуют до запуска команды и, вполне возможно, являются результатами выполнения предыдущих команд. В большинстве операционных систем такими объектами являются записанные в каталоги файлы, которые играют роль интерфейса между командами. Система Оберон расширяет это понятие: ссылки между последовательными командами не ограничиваются файлами, а могут быть любыми глобальными переменными, потому что модули, как сказано выше, не исчезают из памяти по завершении команды.

Такая колоссальная гибкость, похоже, должна открыть ящик Пандоры, и действительно делает это, если неправильно применяется. Причина в том, что состояния глобальных переменных могут целиком определять и менять эффект команды. Переменные представляют скрытые состояния, скрытые в том смысле, что пользователь вообще не знает о них и не имеет никакого простого способа определить их значение. Положительный аспект использования глобальных переменных как интерфейса между командами – в том, что некоторые из них могут быть видны на дисплее. Все окошки вместе с их содержимым сведены в структуру дан-