Классика программирования

Никлаус Вирт, Юрг Гуткнехт

Разработка операционной системы и компилятора

Проект Оберон

ДМК Пресс
издательство

УДК 004.451:004.43Оберон
ББК 32.973.26-018.2
В52

Вирт Н., Гуткнехт Ю.
В52 Разработка операционной системы и компилятора. Проект Оберон: Пер. с англ. Борисов Е.В., Чернышов Л.Н. – М.: ДМК Пресс, 2012. – 560 с.: ил.

ISBN 978-5-94074-672-0

В книге описан проект Оберон, представляющий полную программную среду для современной рабочей станции. Главная цель, поставленная авторами, – спроектировать и реализовать всю систему с нуля и построить ее так, чтобы она могла быть описана, объяснена и понята как единое целое.

В дополнение к основной системе во всех деталях описан компилятор языка Оберон и графическая подсистема.

Для программистов, преподавателей и студентов, изучающих теорию и практику построения операционных систем.

УДК 004.451:004.43Оберон
ББК 32.973.26-018.2

ISBN 0-201-54428-8 (анг.)
ISBN 978-5-94074-672-0 (рус.)

Содержание

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 рассказать, 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 поновому 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 С. – 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 компьютер 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.

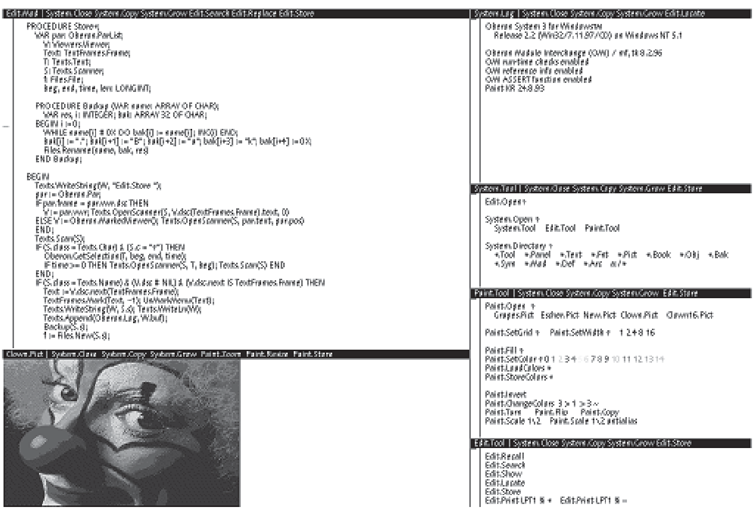

Рис. 2.1. Дисплей Оберона с мозаичными окошками

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 данных, которая задается глобальной переменной в модуле *Viewers*. Таким образом, части этой переменной образуют видимые состояния и вполне уместны в качестве параметров команд.

Таким образом, одно из правил того, что можно назвать стилем программирования Оберона, состоит в том, чтобы избегать скрытых состояний и сокращать количество глобальных переменных. Однако мы не возводим это правило в ранг догмы. Есть поистине полезные исключения, даже если у переменных вообще нет видимых частей.

Остается вопрос: как обозначить видимые объекты в параметрах команды. Очевидный способ – использовать в качестве параметра ближайшее выделение. Процедура определения местонахождения этого выделения обеспечивается модулем *Oberon*. (Она ограничивается выделениями текста.) Другой способ – использовать позицию символа вставки в тексте. Он применяется в случае вставки нового текста; нажатие клавиши на клавиатуре также считается командой и вызывает вставку символов в позицию символа вставки.

Специальное средство – пометка «звездочка» – вводится для указания в качестве операнда окошка. Она помещается в позицию курсора при нажатии на клавиатуре клавиши пометки (SETUP). Процедура *Oberon.MarkedViewer* определяет помеченное «звездочкой» окошко. А команды, для которых оно является параметром, обычно сопровождаются в тексте символом «звездочка». Что именно из помеченного окошка передается в качестве фактического параметра – текст, картинка или любая другая его часть, – зависит от того, как написана процедура команды.

Наконец, не должно остаться незамеченным наиболее приятное свойство системы. Оно – прямое следствие постоянства глобальных переменных и становится ясным, когда команда терпит неудачу. Обнаруженные отказы приводят к прерыванию, которое должно считаться аварийным завершением команды. В худшем случае глобальные данные могут остаться в неопределенном состоянии, но они не будут потеряны, и следующая команда может быть выполнена на их текущем состоянии. Прерывание открывает небольшое окошко со списком вызванных процедур с их локальными переменными и их текущими значениями. Эта информация помогает программисту разобраться в причинах прерывания.

2.2.3. Задачи

Из вышесказанного следует, что система Оберон отличается очень гибкой схемой активации команд. Понятие команды простирается от вставки единственного символа и установки пометки до вычислений, которые могут длиться часами или днями. Кроме того, она отличается очень гибким понятием выбора операнда, не ограничиваясь именованными файлами. И, что более важно, фактическим отсутствием скрытых состояний. Состояние системы практически определяется тем, что видно пользователю.

Это делает ненужным помнить длинную историю ранее активированных команд, запущенных программ, установленных режимов и т. д. На наш взгляд, режимы – характерный признак недружелюбных к пользователю систем. С этой точки зрения становится очевидным, что система позволяет пользователю следить за несколькими различными задачами одновременно. Они представлены в виде окошек, содержащих тексты, графику или иные отображаемые объекты. Пользователь переключается между задачами неявно, выбирая разные окошки в качестве операндов для следующей команды. Суть такой концепции в том, что переключение задач находится под явным контролем пользователя, а атомами действий являются команды.

В то же время мы относим Оберон к однопроцессным (или однопоточным) системам. Как же понять этот очевидный парадокс? Возможно, это лучше всего объяснить, рассматривая основной режим работы. Пока процессор не занят выполнением команды, он непрерывно циклически опрашивает источники событий. Такой цикл называется центральным; он находится в модуле *Oberon*, который можно считать ядром системы. Мышь и клавиатура – два фиксированных источника событий. Если возникает событие клавиатуры, управление передается обработчику, установленному в так называемом окошке-фокусе, который содержит символ вставки. Если возникает событие по нажатию кнопки мыши, управление передается обработчику, в котором в настоящее время находится курсор. Все это возможно под парадигмой единственного непрерывного процесса.

Понятие единственного процесса подразумевает непрерывность, и поэтому эти команды тоже не могут взаимодействовать с пользователем. Взаимодействие ограничено выбором команд перед их выполнением. Следовательно, в типичных программах Оберона нет операторов ввода. Ввод задается параметрами, определенными и подставляемыми до выполнения команды.

Такая схема кажется поначалу довольно ограниченной. Но практически это не так, если рассмотреть действия одного пользователя. Именно он ведет диалог с компьютером. Человек способен вести диалог одновременно с несколькими процессами, только когда отданные им команды являются очень длительными по времени. Мы полагаем, что выполнение длительных вычислений лучше передавать на слабосвязанные серверы вычислений в распределенной системе.

Главное преимущество системы, имеющей дело с единственным процессом, состоит в том, что переключения задач происходят только в определенных пользователем точках, где состояние локального процесса не должно сохраняться до его возобновления. Кроме того, поскольку переключения выбираются пользователем, задачи не могут вносить неожиданные и неконтролируемые помехи при обращении к общим переменным. Поэтому проектировщик системы может опустить все механизмы защиты от таких помех, что существенно упрощает ее.

Основное отличие системы Оберон от многопроцессных систем – в том, что в первой переключения задач происходят только между командами системы, тогда как во вторых переключение может происходить после любой машинной команды. Ясно, что различие – в степени детализации действия. Детализация в Обероне укрупненная, что вполне приемлемо для однопользовательской системы.

Система предоставляет возможность добавления в центральный цикл опрашивающих команд. Это нужно, когда появляются дополнительные источники событий. Известный пример – сеть, в которой команды могут посылаться с других рабочих станций. Центральный цикл просматривает список так называемых дескрипторов задач. Каждый дескриптор ссылается на процедуру команды. Одно из двух стандартных событий – ввод с клавиатуры или нажатие кнопки мыши – происходит, только когда это позволяет их защита (условие). Добавляемые задачи должны иметь свою собственную защиту в начале своих процедур.

При добавлении сетевых команд, именуемых запросами, возникает вопрос: что происходит, когда занятому выполнением другой команды процессору поступает очередной запрос? Очевидно, запрос будет потерян, если не принять меры. Проблема легко исправляется буферизацией ввода. Она есть в любом драйвере устройства ввода – как в драйвере клавиатуры, так и в сетевом драйвере. Входящий сигнал вызывает прерывание, а его обработчик принимает ввод и буферизует его. Подчеркнем, что обработка прерываний – привилегия драйверов, системных компонент самого низкого уровня. Прерывание не вызывает выбора задачи и переключения задач. Управление просто возвращается в точку прерывания, а прерывание остается незаметным для программ. Но у всякого правила есть исключение: при вводе с клавиатуры символа аварийного завершения работы прерывание возвращает управление центральному циклу.

2.2.4. Инструментальные тексты как настраиваемые меню

Конечно, понятия окошек, задающих собственную интерпретацию щелчков мыши, команд, вызываемых из любого текста на экране, любого отображаемого объекта, выбранного в качестве интерфейса между командами, и команд, являющихся недиалоговыми непрерывающимися единицами действий, имеют значительное влияние на стиль программирования в Обероне и полностью меняют стиль использования компьютера. Простота и гибкость, с которыми фрагменты текста могут выделяться, перемещаться, копироваться и обозначать команды и их параметры, резко сокращают необходимость в наборе текста. Мышь становится доминирующим устройством ввода данных, а клавиатура служит только для ввода текстовых данных. Это подкрепляется использованием так называемых инструментальных текстов, наборов часто используемых команд, которые обычно отображаются в узкой системной дорожке окошек. Вам совсем не нужно набирать команды вручную! Обычно они уже видны где-то. Как правило, для любого рабочего проекта пользователь составляет инструментальный текст, который может рассматриваться как собственноручно созданное личное меню.

Редко набираемые на клавиатуре команды имеют еще более приятное достоинство: их имена могут быть осмысленными. Например, операция копирования обозначается как «Copy» вместо «cp», переименования – как «Rename» вместо «rn», выбор каталога файлов именуется «Directory» вместо «ls». Исчезает необходимость в запоминании бесконечного списка загадочных сокращений, что является еще одним признаком недружелюбных к пользователю систем.

Но влияние концепций Оберона не ограничивается только стилем использования компьютера. Оно распространяется и на способ написания программ, взаимодействующих с окружением. В большинстве случаев определение абстрактного типа *Text* в ядре системы предполагает замену файлов текстами как носителями данных ввода и вывода. Получаемое преимущество – это возможность прямого редактирования текста. Например, результат команды *System.Directory* – это требуемый фрагмент каталога файлов в виде (отображаемого) текста. Его часть или он весь могут быть выделены и скопированы в другие тексты обычными командами редактирования (щелчками мыши). Или же компилятор получает в качестве входа те же тексты. Поэтому можно откомпилировать текст, выполнить программу и перекомпилировать исправленный текст без сохранения его на диске между компиляциями и тестированиями. Повсеместная редактируемость текста вместе с наличием глобальных данных (в отдельных окошках) позволяет избежать множества лишних шагов, которые не способствуют прогрессу фактически решаемой задачи.

2.2.5. Расширяемость

Важной целью разработки системы Оберон была расширяемость. Это простота наращивания системы новыми средствами путем добавления модулей, которые могут использовать уже существующие ресурсы. Это (что так же важно) и сокращение системы до тех необходимых средств, которые в настоящее время и на самом деле используются. Например, редактору документов, работающему с документами без рисунков, не нужно загружать обширный графический редактор рабочей станции, работающей автономно, не нужно загружать обширное сетевое программное обеспечение, а офисной системе не нужны ни компилятор, ни ассемблер. Кроме того, система, создающая новый вид кадра дисплея, не должна включать процедуры управления окошками, содержащими такие кадры. Вместо этого она должна использовать существующее управление окошками. Всплески использования памяти многими широко используемыми системами происходят из-за нарушения этих фундаментальных правил разработки. Завышенные требования к памяти для операционной системы хотя и общеприняты, но абсурдны и являются еще одним признаком недружелюбия к пользователю или, возможно, признаком дружелюбия между производителями. Причина тому – не что иное, как неадекватная расширяемость.

Мы не ограничиваем это понятие процедурной расширяемостью, которую просто реализовать. Важно то, что расширение – это не только добавление новых процедур и функций, но введение собственных типов данных, построенных на базисе, обеспеченном системой, то есть расширяемость данных. Например, графическая система должна быть в состоянии определить свои графические кадры на базе кадров, обеспеченных основным модулем отображения, путем расширения их необходимыми графическими атрибутами.

Это требует адекватных языковых средств. Язык Оберон обеспечивает такие средства в виде расширений типов. Язык был создан именно по этой причине. (Язык Модула-2 был бы таковым, если бы не отсутствие в нем этих возможностей.) Его влияние на структуру системы было глубоким, а результаты были самыми обнадеживающими. Например, многие дополнения были созданы с удивительной легкостью. Одно из них описано в конце этой книги. При этом базовая система весьма скромна в своих требованиях к ресурсам (см. таблицу в конце раздела 2.3).

2.2.6. Динамическая загрузка

Активация команд, находящихся в модулях, которые не загружены в память, предполагает загрузку модулей и, конечно, всех их импортов. Вызов загрузчика, однако, не ограничивается активацией команды; он может также осуществляться программными обращениями к процедурам. Такая возможность необходима для успешной реализации подлинной расширяемости. Модули должны загружаться по требованию. Например, редактор документов загружает графический пакет, только если в активном документе есть графический элемент, но не иначе.

В системе Оберон нет отдельного компоновщика. Модуль связывается с его импортами при загрузке, но не ранее. Поэтому каждый модуль представлен только одним экземпляром как в основной памяти (связанный), так и во вспомогательной (несвязанный, в виде файла). Уход от тиражирования копий в разных связанных объектных файлах – это ключ к экономии памяти. Заранее связанных мегафайлов в системе Оберон нет, а каждый модуль – повторно используемый.

2.3. Структура системы

Наибольшая опознаваемая единица системы – модуль. Поэтому и при описании системы наиболее уместен модульный подход. А так как интерфейсы модулей явно объявлены, их взаимозависимость тоже легко показать в виде орграфа, дуги которого суть импорты.

Граф модулей системы Оберон – иерархический, то есть не имеет циклов. Самые нижние члены иерархии фактически импортируют только аппаратные средства. Здесь имеются в виду модули, содержащие драйверы устройств. Хотя модуль *Kernel* тоже относится к этому классу: он «импортирует память» и включает драйвер диска. Модули на вершине иерархии фактически экспортируются пользователю. Поскольку пользователь имеет прямой доступ к процедурам команд, мы называем этих высших членов иерархии командными, или инструментальными, модулями.

Иерархия базовой системы показана на рис. 2.2. Граф упрощен за счет исключения дуг непосредственного импорта, когда косвенный путь тоже ведет от источника к приемнику. Например, *Files* импортирует *Kernel*; прямой импорт не показан, потому что путь от *Kernel* ведет к *Files* через *FileDir*. Имена модулей *во множественном числе* обычно указывают на определение абстрактного типа данных в модуле. Тип экспортируется вместе с его операциями. Примеры – *Files*, *Modules*, *Fonts*, *Texts*,*Viewers*, *MenuViewers* и *TextFrames*. (Исключение – вспомогательный модуль *Reals* для *Texts*, содержащий операции преобразования чисел с плавающей запятой, присутствующих в коде ассемблера.) Модули с именами *в единственном числе* обычно обозначают управляемый модулем ресурс, будь то