

Глава 10. Понятие о системе программирования

Сейчас компьютер позволяет решать все те проблемы, которые до его изобретения не существовали, и, следовательно, их и не требовалось решать.

Мысли о прогрессе...

Суть дела не в полноте знания, а в полноте разума.

Демокрит, VI век до н.э.

Как уже упоминалось, все программы, которые выполняются на компьютере, можно разделить на два вида – *прикладные* и *системные*. Вообще говоря, компьютеры существуют в основном для того, чтобы выполнять прикладные программы, однако понятно, что в данной книге нас в первую очередь будет интересовать не прикладное, а именно системное программирование.

Все системные программы можно, в свою очередь, тоже разделить на две части. В одну часть входят программы, предназначенные для управления оборудованием ЭВМ (например, так называемые драйверы), а также программы, управляющие на компьютере выполнением *других* программ. Кроме того, обычно сюда же включают и служебные программы для управления обрабатываемыми данными (так называемую файловую систему). Программы этого класса входят в большой комплекс системных программ, называемый **операционной системой** ЭВМ, эта тема должна изучаться в отдельном курсе.

В другую часть входят системные программы, предназначенные для автоматизации процесса разработки, создания, модификации и эксплуатации программ. Эти программы входят в состав **системы программирования**. Не надо, однако, думать, что система программирования состоит только из таких системных программ, которые помогают писать новые программы. Система программирования является *комплексом*, в состав которого входят языковые, программные и информационные компоненты. Ниже перечислены все основные компоненты, входящие в систему программирования.

10.1. Компоненты системы программирования

Программист должен обладать способностью первоклассного математика к абстракции в сочетании с эдисоновским талантом соорудить всё, что угодно, из нуля и единицы. Он должен сочетать аккуратность бухгалтера с проницательностью разведчика, фантазию автора детективных романов – с трезвой практичностью бизнесмена. А, кроме того, программист должен иметь вкус к коллективной работе, понимать интересы пользователя и многое другое...

Андрей Петрович Ершов¹.

1. **Языки системы программирования.** Сюда относятся как языки программирования, предназначенные для записи алгоритмов (C, Java, Python Паскаль, Фортран, Ассемблер и т.д.), так и другие языки, которые служат для управления самой системой программирования, например, так называемый командный язык (язык командных файлов). Другие языки, входящие в систему программирования, могут предназначаться для автоматизации разработки больших программ (например, так называемый язык спецификации программ). Вы не должны при этом путать три разных понятия: **язык** (например, Ассемблер), **программу** на этом языке и **компилятор**, который переводит Ассемблерные программы (на объектный язык).

¹ Один из основоположников отечественного теоретического и системного программирования, разработчик первых трансляторов, и, между прочим, друг Дональда Кнута.

2. **Служебные программы системы программирования.** Со многими из этих программ Вы уже познакомились в этой книге, например, сюда входят такие программы.
 1. **Текстовые редакторы**, предназначенные для набора и исправления текстов программ на языках программирования (обычно это исходные модули).



Существуют языки программирования, которые требуют специальных текстовых редакторов. Например, вот программа знаменитой игры «Жизнь» на старом функциональном языке сверхвысокого уровня APL:

```
life ← { ↑1 ω∨.Λ3 4=+/,~1 ∅ 1ο.Θ~1 ∅ 1ο.Φ⊂ω }
```

Обратите внимание, как символ нуля $\emptyset$ отличается от символов Θ , $\circ$ и Φ . Ясно, что текст таких программ надо набирать в особом редакторе.

2. **Трансляторы** (компиляторы и интерпретаторы) для перевода с одного языка на другой. При этом **компиляторы** целиком переводят исходный модуль (например, компилятор Ассемблера переводит исходный модуль с языка Ассемблер на язык объектных модулей). Часто компиляторы могут переводить с языка высокого уровня на Ассемблер. **Интерпретаторы** производят построчный (точнее, пооператорный) перевод исходного текста программы (на язык машины или на некоторый промежуточный язык) и тут же выполняют полученный оператор.



Заметим, что если исходные модули (жаргонное название – «исходники») программист может легко изменять с помощью текстового редактора, то объектные модули изменить практически нельзя (их можно только получить заново из изменённых исходных модулей). Более точно, попытка изменения объектного модуля выливается в задачу программирования на языке машины (не на Ассемблере!), и что бы там не говорили «крутые» программисты, никакие Дизассемблеры при модификации достаточно большой программы кардинально помочь не могут.

Именно поэтому фирмы, продающие своё программное обеспечение, поставляют его пользователям чаще всего в виде загрузочных и объектных модулей, и пуще глаза берегут исходные тексты программ. В то же время существует и фонд свободного программного обеспечения FSF (Free Software Foundation), основанная в 1985 году Ричардом Столманом (Richard Stallman), этот фонд пропагандирует свободное распространение программных продуктов и публикацию исходных текстов программ.

3. **Редакторы внешних связей**, собирающие загрузочный модуль из объектных модулей.
4. **Загрузчики**, запускающие программы на счёт.
5. **Отладчики**, помогающие пользователям в диалоговом режиме искать и исправлять ошибки в своих программах.



Различают отладчики программ на языке машины (в кодах машины) и так называемые отладчики в терминах исходного языка программирования. Для отладчиков последнего вида при компиляции модулей с исходного языка в паспортах объектных модулей сохраняется достаточно полная информация о *локальных* именах, использованных пользователем (это имена меток, процедур, переменных, констант и т.д.). В дальнейшем эта информация переносится редактором внешних связей в паспорт загрузочного модуля, и может быть использована таким отладчиком при выдаче сообщений об ошибках (например, «При выполнении третьего оператора в процедуре Summa деление на переменную X, равную нулю»).

6. **Профилировщики** (профайлеры), которые определяют, какой процент времени выполняется та или иная часть программы. Это позволяет выявить наиболее интенсивно используемые фрагменты программы, так называемые горячие точки (Hot Spots), и оптимизировать их на исходном языке или, например, переписать эти фрагменты на Ассемблере.
7. **Библиотекари**, которые позволяют создавать и изменять файлы-библиотеки процедур lib и dll.
8. И другие служебные программы.
3. **Информационное обеспечение системы программирования.** Сюда относятся различные структурированные описания языков, служебных программ, библиотек модулей и т.п. Без хорошего информационного обеспечения современные системы программирования эффективно работать не могут. Каждый пользователь неоднократно работал с этой компонентой системы

программирования, нажимая функциональную клавишу F1, выбирая из меню пункт Help (Помощь) или извлекая информацию из самой программы (например, `Link /?`).

На рис. 10.1 показана общая схема прохождения программы пользователем через систему программирования. Программные модули пользователя на этом рисунке заключены в прямоугольники, а системные (служебные) программы – в прямоугольники с закруглёнными углами. На этой схеме можно проследить весь путь, по которому проходит программа от написания её текста на некотором языке программирования, до этапа счёта.

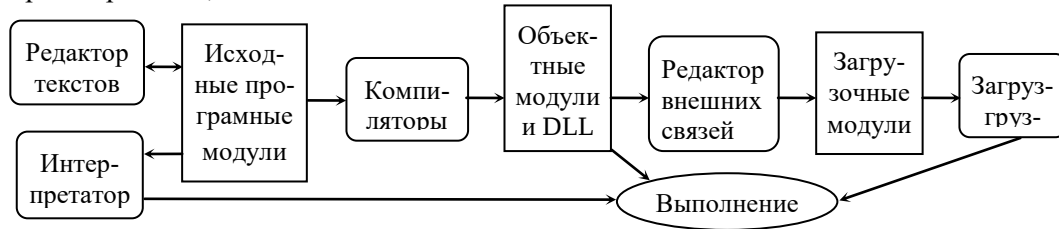


Рис. 10.1. Общая схема прохождения программы через систему программирования.

Заметим, что сейчас для многих языков программирования созданы так называемые *интегрированные среды разработки* (IDE – Integrated Development Environment), включающие в себя работающие под общим управлением почти все компоненты системы программирования. Примером такой интегрированной среды разработки программного обеспечения является знакомая программистам система Free Pascal.

На этом закончим описание состава системы программирования и переходим к рассмотрению характеристик исполняемых программ.

10.2. Характеристики исполняемых программ

Продолжайте учиться. Узнайте больше о компьютерах, ремеслах, садоводстве, о чем угодно. Никогда не оставляйте мозг в лености.

Джордж Карлин¹



Модульная программа состоит из отдельных модулей, которые могут обладать некоторыми специфическими характеристиками, к рассмотрению которых нам сейчас и надо перейти.

10.2.1. Повторно выполняемые программы

Бывают ошибки, которые очень хочется повторить.

Веселин Георгиев,
юморист и творец афоризмов

Повторное выполнение программы предполагает, что, будучи один раз загруженной в оперативную память, она допускает своё многократное исполнение (т.е. вход в начало этой программы после её завершения). Естественно, что процедуры и функции по самому их смыслу являются повторно используемыми, однако для основной (головной) программы дело обстоит сложнее. Какими свойствами должна обладать программа на Ассемблере, чтобы после, например, после окончания счёта по макрокоманде `exit` было возможно снова войти в начало программы по метке `main`, не загружая эту программу заново в оперативную память?

Естественно, что, прежде всего, необходимо восстановить для программы первоначальный (пустой) стек, закрыть ненужные файлы и заново подключить стандартные потоки ввода/вывода. Этого, однако, может оказаться недостаточно. Легко понять, что для головной программы на Ассемблере должно выполняться следующее условие: программа не должна менять свои кодовые сегменты и переменные с начальными значениями в сегментах данных, или, в крайнем случае, восстанавливать эти

¹ Американский стендап-комик 🤪, актёр, писатель, сценарист, продюсер, обладатель четырёх премий «Грэмми» и премии Марка Твена. Автор 5 книг, снялся в 16 фильмах.

значения перед окончанием программы. Например, если в программе на Ассемблере имеются предложения

```
X dw 1
. . .
mov X, 2
```

то программа будет повторно используемой только тогда, когда она восстанавливает первоначальное значение переменной X перед выходом из программы. В настоящее время это свойство программы не имеет большого значения, и важным является более сильное свойство программы – быть *повторно-входимой* (**реентерабельной**).

10.2.2. Повторно-входимые (реентерабельные) программы

Нельзя войти в одну и ту же реку дважды.

Гераклит Эфесский, V век до н.э.

Можно ли войти в одну и ту же реку дважды, не выйдя из неё в первый раз ? 🤔

Свойство исполняемой программы быть реентерабельной (reentrant, по-русски – повторно-входимой, иногда говорят – параллельно используемой), является очень важным, особенно при написании системных программ. Программа называется реентерабельной, если она допускает повторный вход в своё начало до выхода из этой программы.

Особо подчеркнем, что повторный вход в такую программу (т.е. порождение нового вычислительного потока) производит *не сама* эта программа, используя прямую или косвенную рекурсию (в этом случае говорят о рекурсивном вызове подпрограммы), а *другие программы* (например, диспетчер задач операционной системы). Таким образом, внутри реентерабельной программы могут располагаться *несколько* текущих точек выполнения этой программы, что и означает одновременное существование несколько вычислительных процессов. Такая ситуация уже встречалась при изучении системы прерываний, когда выполнение процедуры-обработчика прерывания с некоторым номером могло быть прервано новым сигналом прерывания с таким же номером, так что производился *повторный вход* в начало *этой же* процедуры до окончания обработки текущего прерывания.

Каждый поток реентерабельной программы имеет, как уже упоминалось, своё *поле сохранения* (или *контекст*). При прерывании выполнения процесса там сохраняются, в частности, все регистры, определяющие текущую точку выполнения. Это как сегментные регистры, так и регистры общего назначения, а также системные и управляющие регистры.

Главное отличие реентерабельных программ от обычных *рекурсивных* подпрограмм заключается именно в том, что, в отличие от вызова программы, при *каждом входе* в реентерабельную программу порождается новый вычислительный поток и, соответственно, новый TSS со своим дескриптором в дескрипторной таблице GDT. Это позволяет продолжить выполнение реентерабельной программы с *любой* из этих нескольких текущих точек выполнения программы, восстановив значения всех её регистров (контекста) для этой точки программы.

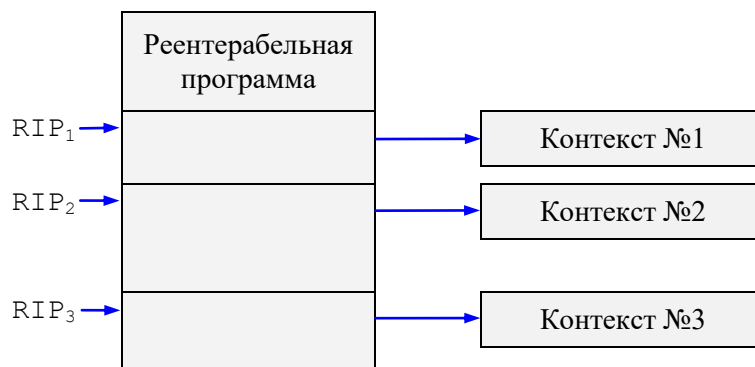


Рис. 10.2. Реентерабельная программа с тремя точками выполнения.

На рис. 10.2 показан пример реентерабельной программы с тремя точками выполнения, отмеченными значениями регистра счётчика адреса RIP_i. Какие-нибудь из этих точек в данный момент могут быть активными, т.е. в них процессоры (процессорные ядра) выполняют команды программы.

Возможен, однако, и случай, когда все эти три вычислительных процесса находятся в состоянии *ожидания*, а процессорные ядра заняты выполнением каких-либо других потоков.

С точки зрения программиста все вычислительные потоки реентерабельной программы имеют один общий сегмент кода, но свои сегменты данных и стека.

Повторный вход в начало программы можно выполнить двумя способами. Во-первых, приложение может попросить диспетчер задач породить новую задачу (в терминах Windows) или процесс (в ОС Unix). У новой задачи будет тот же самый сегмент кода, но свой сегмент данных (копия сегмента старой задачи), и свой (пустой) стек.



Не надо понимать слова «имеют общие сегменты данных» слишком буквально. Например, сначала у разных задач сегменты данных тоже общие (а не копии), однако, когда один процесс пишет что-нибудь в свой сегмент данных, то данная страница сегмента дублируется и уже будет у двух процессов *разной*. Таким образом, копируются только *различающиеся* страницы сегментов данных. Этот механизм называется «копирование при записи». Свои (полностью личные) у задач только сегменты стека.

Во-вторых, и сама задача может породить новый вычислительный процесс.¹ У нового процесса будет тот же самый сегмент кода и копии сегментов данных и стека. Обычно, чтобы избежать копирования данных, свои «личные» данные процессы хранят не в сегменте данных, а в динамических переменных (в куче) или в своих стеках. Таким образом, одна реентерабельная программа, находясь в памяти ЭВМ, может породить не один, а несколько самостоятельных (независимых друг от друга) *вычислительных процессов*.

Хорошим примером реентерабельной программы может служить текстовый процессор WORD, который порождает отдельную задачу на каждый редактируемый документ, и на некоторые операции. Так, операция поиска выполняется своей задачей.

Переключения между потоками одной задачи происходят много быстрее, чем между «независимыми» задачами, так как контекст потока меньше контекста задачи. Некоторые языки (например, Java) для приложения (программы) не могут порождать «настоящие» параллельные задачи, а только потоки.

Принцип выполнения реентерабельных программ можно пояснить на таком шутливо-детективном примере. Вернувшись из очередного краткого отпуска, следователь по особо важным делам майор Пронин принял к производству дело опасного преступника Х. Произведя первые допросы свидетелей, и приобщив их к материалам дела, Пронин был вынужден прервать данное расследование, так как его попросили срочно заняться новым преступлением, которое предположительно совершил вор-рецидивист У. Найдя в новом деле первые улики и отправив их на экспертизу, которая должна была занять определённое время, Пронин вернулся к делу преступника Х, но вскоре после этого ему позвонил его начальник полковник Петренко и приказал немедленно заняться делом серийного убийцы Z... Стоит пожалеть майора Пронина и разрешить ему хотя бы в воскресенье поехать на дачу и совсем не заниматься расследованиями 😊.



Предполагается, что, подобно большинству людей, майор Пронин не может вести два дела *по настоящему* одновременно, подобно тому, как компьютер, обладающий только одним центральным процессором, в каждый момент времени может выполнять только одну программу. Психологи говорят, что людей, которые могут на самом деле одновременно заниматься несколькими разными делами, очень мало. По преданию, одним из них был знаменитый древнеримский император Гай Юлий Цезарь, который мог слушать доклад одного чиновника, отдавать приказание другому и одновременно писать письмо по совершенно иной теме. Заметим, что среди компьютеров массового производства до широкого распространения многоядерных процессоров тоже было мало многопроцессорных ЭВМ. Таким образом мы считаем, что наш майор Пронин имеет в голове только один «процессор».

Как видим, следователь Пронин может возобновить расследование любого порученного ему дела, освежив свою память материалами из соответствующей папки (или из компьютерного файла, если майор Пронин шагает в ногу со временем). Точно так же и выполнение любого потока реентерабельной программы можно продолжить с прерванного места, так как вся необходимая для продолжения

¹ В системах семейства Unix для этого служит знаменитый системный вызов `fork()`, Вам предстоит познакомиться с ним в курсе по операционным системам.

работы информация находится в своей области сохранения. Таким образом, в нашей аналогии майор Пронин играет роль центрального процессора, а дела, которые он ведет – это вычислительные потоки одной общей реентерабельной «задачи расследования преступлений».

В нашем примере на рис. 10.2 для служебной программы, управляющей счётом задач (эта программа операционной системы чаще всего называется планировщиком процессов или *диспетчером задач*) (task scheduler), для одной копии реентерабельной программы в памяти будут существовать три независимых единицы работы – три вычислительных потока. Разумеется, программа может порождать и всего одну задачу и один вычислительный поток.

Как можно заметить, каждый вычислительный поток, независимо от того, как он был порожден, может находиться в одном из следующих состояний:

- Поток выполняется в процессоре (майор Пронин допрашивает серийного убийцу Z).
- Поток готов к счёту, но процессор занят (преступник X сидит в КПЗ, майор Пронин не вызывает его на допрос, так как занят другим расследованием).
- Поток не готов к счёту и чего-то ждёт (ожидаются результаты экспертизы по делу вора-рецидивиста Y и майор Пронин и не вызывает его на допрос).
- Поток завершен, но ещё не удален из списка процессов диспетчера задач (производство по делу завершено, но само дело ещё не направлено в суд).



В операционных системах Unix такие закончившиеся процессы носят меткое название «зомби», они, хотя и завершены, но «не совсем мертвы» и не удаляются из системы до тех пор, пока некоторые из ещё выполняющихся процессов (в частности, их родительская задача) могут «поинтересоваться» их судьбой. Например, один поток может запросить, закончился ли другой поток нормально или аварийно, сколько он использовал ресурсов и т.д. А иногда у ОС просто ещё «не дошли руки», чтобы удалить TSS этого потока из таблицы дескрипторов.

На рис. 10.3 показана диаграмма перехода между различными состояниями вычислительного потока во время его выполнения.

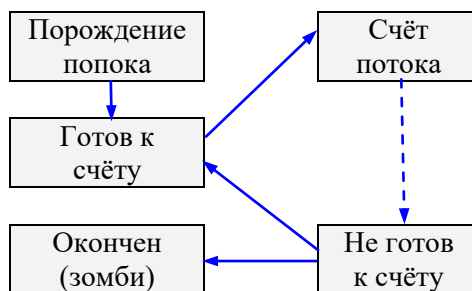


Рис. 10.3. Диаграмма перехода между состояниями потока:
 ————— диспетчер задач; - - - - - прерывание.

Ниже перечислены основные свойства, которыми должна обладать программа на Ассемблере, чтобы быть реентерабельной.

- Программа не изменяет свои сегменты кода (обычно они просто закрыты на запись).
- Программа либо совсем не имеет собственных сегментов данных (т.е. все данные передаются ему в стеке в качестве параметров), либо при каждом входе получает *копии* своих сегментов данных (не забываем механизм «копирование при записи»).
- При каждом входе в программу оно получает новый сегмент стека, пустой для основной программы и с фактическими параметрами и адресом возврата для процедуры.



Дальнейшее развитие программирования привело к появлению уже «облегчённых» процессов, которые называются *волокна* (fibber), им запрещено делать системные вызовы (в частности, работать с файлами и внешними устройствами), т.е. они всегда выполняются в режиме пользователя. Разным волокнам одного потока запрещено параллельно выполняться на разных кпроцессорных ядрах (только на одном ядре и по-очереди, причёт очерёдность счёта волокон определяет сам поток). В ОС Windows у каждой задачи есть хотя бы один поток, а у каждого потока есть хотя бы одно волокно. Волокна создавать проще, чем потоки но в остальном они не имеют перед ними особых преимуществ и используются достаточно редко.

Реентерабельность является особенно важной при написании программ, входящих в состав операционных систем и систем программирования. Это следует из того, что если некоторая системная

программа (например, компилятор с Ассемблера или DLL-библиотека) является реентерабельной, то в оперативной памяти достаточно иметь только одну копию этой программы, которая может одновременно использоваться для любого числа программ (отсюда второе название таких программ – параллельно используемые).¹



Как можно заметить, переключение с одного потока на другой занимает довольно много времени из-за относительно большого объема данных, которые необходимо запомнить и восстановить, используя области сохранения. Вот и наш майор Пронин, чтобы переключиться с одного дела на другое должен предпринять целый ряд действий. Он должен выключить видеокамеру, собрать разложенные на столе документы и видеоматериалы в папку (это контекст дела!), спрятать её в сейф, вызвать конвой и отправить подследственного в камеру. После этого он должен вызвать на допрос нового подследственного, достать из сейфа его папку, разложить на столе необходимые документы, подготовить записывающую аппаратуру и т.д.

Ясно, что при большом количестве одновременно проводимых расследований и при частом переключении с одного дела на другое всё больше времени следователя будет тратиться непродуктивно. Для решения этой проблемы начальство выделило Пронину для работы сразу несколько одинаковых тщательно охраняемых расположенных рядом кабинетов, в каждом из которых есть всё для работы: стол с документами, видеокамера, сейф, подследственный на стуле с конвоем и т.д. Теперь переключение с одного дела на другое заключается для Пронина в быстром переходе из одного кабинета в другой, где уже всё готово для немедленного продолжения работы со следующим подозреваемым.

Аналогично, для быстрого переключения с одного потока на другой в архитектуре современных компьютеров у каждого процессорного ядра делается несколько одинаковых наборов необходимых регистров (так называемый регистровый файл, не надо путать его с набором теневых регистров для работы конвейера, о чем будет рассказано в другой главе). Совокупность всех регистров (как регистров программиста, т.е. RAX, RBX и т.д., так и набора служебных регистров RIP, RFLAGS и т.д.) определяет текущую точку выполнения задачи, поэтому, производя смену набора всех регистров, фактически производится переключение с одного потока на другой. Такая технология приводит к тому, что на одном процессоре как бы реализуется несколько *виртуальных* процессоров (виртуальных процессорных ядер со своими наборами всех регистров и своими контроллерами внутренних прерываний). Такой способ работы называется гиперпоточностью (Hyper Threading), он реализован на некоторых современных ЭВМ.

В современных ЭВМ большинство системных программ являются реентерабельными.

Вопросы и упражнения

*Принимаясь за дело, соберись с духом.
Козьма Прутков*

1. Каково назначение системы программирования ?
2. Из каких главных компонентов состоит система программирования ?
3. Чем отличаются компиляторы от интерпретаторов ?
4. Что такое повторно-выполнимые модули, и какие ограничения необходимо наложить на Ассемблерную программу, чтобы она была повторно-выполнимой ?
5. Какие исполняемые программы называются повторно-входимыми (реентерабельными) ?
6. Какие требования необходимо наложить на Ассемблерную программу, чтобы она была реентерабельной ?
7. Для чего системные программы делают реентерабельными ?
8. Что такое вычислительный процесс и как одна программа может порождать несколько вычислительных процессов ?

¹ Естественно, что на однопроцессорной (одноядерной) ЭВМ в каждый момент времени может компилироваться только одна программа, но, как вскоре будет описано, по сигналам прерывания от внутренних часов компьютера, можно производить быстрое переключение с одного вычислительного потока на другой, так что создаётся впечатление, что компилируются (хотя и более медленно) несколько программ сразу.

9. Ещё раз прочитайте приведённое выше шуточное сравнение выполнения реентерабельной программы с деятельностью следователя майора Пронина и ответьте на следующий вопрос. Что должен делать процессор, когда его аналог майор Пронин отдыхает на своей даче и не занят расследованием преступлений? Подсказка: как обычно в детективах, отдых майора Пронина (а что это в нашем компьютере?) в любой момент может быть нарушен неожиданным звонком об очередном событии, требующим внимания майора Пронина.