**Системное программирование**

**Системная программа** – программа, предназначенная для поддержания работоспособности СОИ или повышения эффективности ее использования.

**Прикладная программа** – программа, предназначенная для решения задачи или класса задач в определенной области применения СОИ.

В соответствии с терминологией, системное программирование — это процесс разработки системных программ (в том числе, управляющих и обслуживающих).

С другой стороны, система — единое целое, состоящее из множества компонентов и множества связей между ними. Тогда системное программирование — это разработка программ сложной структуры.

Эти два определения не противоречат друг другу, так как разработка программ сложной структуры ведется именно для обеспечения работоспособности или повышения эффективности СОИ.

Подразделение ПО на системное и прикладное является до некоторой степени устаревшим. Сегодняшнее деление предусматривает по меньшей мере три градации ПО:

* Системное
* Промежуточное
* Прикладное

Промежуточное ПО (middleware) мы определяем, как совокупность программ, осуществляющих управление вторичными (конструируемыми самим ПО) ресурсами, ориентированными на решение определенного (широкого) класса задач. К такому ПО относятся менеджеры транзакций, серверы БД, серверы коммуникаций и другие программные серверы. С точки зрения инструментальных средств разработки промежуточное ПО ближе к прикладному, так как не работает на прямую с первичными ресурсами, а использует для этого сервисы, предоставляемые системным ПО.

С точки зрения алгоритмов и технологий разработки промежуточное ПО ближе к системному, так как всегда является сложным программным изделием многократного и многоцелевого использования и в нем применяются те же или сходные алгоритмы, что и в системном ПО.

Современные тенденции развития ПО состоит в снижении объема как системного, так и прикладного программирования. Основная часть работы программистов выполняется в промежуточном ПО. Снижение объема системного программирования определено современными концепциями ОС, объектно-ориентированной архитектурой и архитектурой микроядра, в соответствии с которыми большая часть функций системы выносится в утилиты, которые можно отнести и к промежуточному ПО. Снижение объема прикладного программирования обусловлено тем, что современные продукты промежуточного ПО предлагают все больший набор инструментальных средств и шаблонов для решения задач своего класса.

Значительная часть системного и практически все прикладное ПО пишется на языках высокого уровня, что обеспечивает сокращение расходов на их разработку/модификацию и переносимость.

Системное ПО подразделяется на системные управляющие программы и системные обслуживающие программы.

**Управляющая программа** — системная программа, реализующая набор функций управления, который включает в себя управление ресурсами и взаимодействие с внешней средой СОИ, восстановление работы системы после проявления неисправностей в технических средствах.

**Программа обслуживания (утилита)** — программа, предназначенная для оказания услуг общего характера пользователям и обслуживающему персоналу СОИ.

Управляющая программа совместно с набором необходимых для эксплуатации системы утилит составляют **операционную систему** (ОС).

Кроме входящих в состав ОС утилит могут существовать и другие утилиты (того же или стороннего производителя), выполняющие дополнительное (опционное) обслуживание. Как правило, это утилиты, обеспечивающие разработку программного обеспечения для операционной системы.

**Система программирования** — система, образуемая языком программирования, компилятором или интерпретатором программ, представленных на этом языке, соответствующей документацией, а также вспомогательными средствами для подготовки программ к форме, пригодной для выполнения.

**Этапы подготовки программы**

При разработке программ, а тем более — сложных, используется принцип модульности, разбиения сложной программы на составные части, каждая из которых может подготавливаться отдельно. Модульность является основным инструментом структурирования программного изделия, облегчающим его разработку, отладку и сопровождение.

**Программный модуль** — программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

При выборе модульной структуры должны учитываться следующие основные соображения:

* **Функциональность** — модуль должен выполнять законченную функцию
* **Несвязность** — модуль должен иметь минимум связей с другими модулями, связь через глобальные переменные и области памяти нежелательна
* **Специфицируемость** — входные и выходные параметры модуля должны четко формулироваться

Программа пишется в виде исходного модуля.

**Исходный модуль** — программный модуль на исходном языке, обрабатываемый транслятором и представляемый для него как целое, достаточное для проведения трансляции.

Первым (не для всех языков программирования обязательным) этапом подготовки программы является обработка ее Макропроцессором (или Препроцессором). Макропроцессор обрабатывает текст программы и на выходе его получается новая редакция текста. В большинстве систем программирования Макропроцессор совмещен с транслятором, и для программиста его работа и промежуточный ИМ «не видны».

Следует иметь в виду, что Макропроцессор выполняет обработку текста, это означает, с одной стороны, что он «не понимает» операторов языка программирования и «не знает» переменных программы, с другой, что все операторы и переменные Макроязыка (тех выражений в программе, которые адресованы Макропроцессору) в промежуточном ИМ уже отсутствуют и для дальнейших этапов обработки «не видны».

Так, если Макропроцессор заменил в программе некоторый текст A на текст B, то транслятор уже видит только текст B, и не знает, был этот текст написан программистом «своей рукой» или подставлен Макропроцессором.

Следующим этапом является трансляция.

**Трансляция** — преобразование программы, представленной на одном языке программирования, в программу на другом языке программирования, в определенном смысле равносильную первой.

Как правило, выходным языком транслятора является машинный язык целевой вычислительной системы. (Целевая ВС — та ВС, на которой программа будет выполняться.)

**Машинный язык** — язык программирования, предназначенный для представления программы в форме, позволяющей выполнять ее непосредственно техническими средствами обработки информации.

**Трансляторы** — общее название для программ, осуществляющих трансляцию. Они подразделяются на Ассемблеры и Компиляторы — в зависимости от исходного языка программы, которую они обрабатывают. Ассемблеры работают с Автокодами или языками Ассемблера, Компиляторы — с языками высокого уровня.

**Автокод** — символьный язык программирования, предложения которого по своей структуре в основном подобны командам и обрабатываемым данным конкретного машинного языка.

**Язык высокого уровня** — язык программирования, понятия и структура которого удобны для восприятия человеком.

**Объектный модуль** — программный модуль, получаемый в результате трансляции исходного модуля.

Поскольку результатом трансляции является модуль на языке, близком к машинному, в нем уже не остается признаков того, на каком исходном языке был написан программный модуль. Это создает принципиальную возможность создавать программы из модулей, написанных на разных языках. Специфика исходного языка, однако, может сказываться на физическом представлении базовых типов данных, способах обращения к процедурам/функциям и т.п. Для совместимости разноязыковых модулей должны выдерживаться общие соглашения. Большая часть объектного модуля — команды и данные машинного языка именно в той форме, в какой они будут существовать во время выполнения программы. Однако, программа в общем случае состоит из многих модулей. Поскольку транслятор обрабатывает только один конкретный модуль, он не может должным образом обработать те части этого модуля, в которых запрограммированы обращения к данным или процедурам, определенным в другом модуле. Такие обращения называются внешними ссылками. Те места в объектном модуле, где содержатся внешние ссылки, транслируются в некоторую промежуточную форму, подлежащую дальнейшей обработке. Говорят, что объектный модуль представляет собой программу на машинном языке с неразрешенными внешними ссылками. Разрешение внешних ссылок выполняется на следующем этапе подготовки, который обеспечивается Редактором Связей (Компоновщиком). Редактор Связей соединяет вместе все объектные модули, входящие в программу. Поскольку Редактор Связей «видит» уже все компоненты программы, он имеет возможность обработать те места в объектных модулях, которые содержат внешние ссылки. Результатом работы Редактора Связей является загрузочный модуль.

**Загрузочный модуль** — программный модуль, представленный в форме, пригодной для загрузки в оперативную память для выполнения.

Загрузочный модуль сохраняется в виде файла на внешней памяти. Для выполнения программа должна быть перенесена (загружена) в оперативную память. Иногда при этом требуется некоторая дополнительная обработка (например, настройка адресов в программе на ту область оперативной памяти, в которую программа загрузилась). Эта функция выполняется Загрузчиком, который обычно входит в состав операционной системы. Возможен также вариант, в котором редактирование связей выполняется при каждом запуске программы на выполнение и совмещается с загрузкой. Это делает Связывающий Загрузчик. Вариант связывания при запуске более расходный, т.к. затраты на связывание тиражируются при каждом запуске. Но он обеспечивает:

* большую гибкость в сопровождении, так как позволяет менять отдельные объектные модули программы, не меняя остальных модулей;
* экономию внешней памяти, т.к. объектные модули, используемые во многих программах не копируются в каждый загрузочный модуль, а хранятся в одном экземпляре.

Вариант интерпретации подразумевает прямое исполнение исходного модуля.

**Интерпретация** — реализация смысла некоторого синтаксически законченного текста, представленного на конкретном языке.

Интерпретатор читает из исходного модуля очередное предложение программы, переводит его в машинный язык и выполняет. Все затраты на подготовку тиражируются при каждом выполнении, следовательно, интерпретируемая программа принципиально менее эффективна, чем транслируемая. Однако, интерпретация обеспечивает удобство разработки, гибкость в сопровождении и переносимость.

Не обязательно подготовка программы должна вестись на той же вычислительной системе и в той же операционной среде, в которых программа будет выполняться. Системы, обеспечивающие подготовку программ в среде, отличной от целевой называются кросс-системами. В кросс-системе может выполняться вся подготовка или ее отдельные этапы:

* Макрообработка и трансляция
* Редактирование связей
* Отладка

Типовое применение кросс-систем — для тех случаев, когда целевая вычислительная среда просто не имеет ресурсов, необходимых для подготовки программ, например, встроенные системы. Программные средства, обеспечивающие отладку программы на целевой системе можно также рассматривать как частный случай кросс-системы.