зовом проблемной задачи загружает все необходимые служебные модули, обращается к подпрограмме IBCOM# со смещением 64 для установления фортранонской среды, после чего передает управление головной подпрограмме проблемной задачи, сообщая ей массив с адресами загруженных системных подпрограмм. Головная подпрограмма загружает необходимые проблемные модули, как было описано выше, и передает в подпрограмму, определяющую алгоритм задачи, адреса загрузки системных и прикладных модулей. Далее необходимо обеспечить, чтобы адреса загрузки системных модулей попали в поля памяти, где расположены адресные константы этих системных модулей, неопределенные на этапе редактирования. Сделать это можно, например, следующим образом. Учитывая, что память под массивы отводится в модуле вслед за памятью, занятой под адресные константы, в этом модуле резервируется массив из одного элемента. Зная из протокола трансляции порядок расположения адресных констант и используя элементы введенного массива с отрицательным смещением, адрес загрузки нужного системного модуля помещается на место соответствующей адресной константы.

Следует отметить, что такая схема динамического связывания помимо экономии оперативной памяти обеспечивает вывод информации из отдельно редактированных модулей в один набор данных.

Описанная схема организации динамической структуры программ была использована при разработке диалоговой системы анализа и планирования ДИСПЭК-1 [3].

ЛИТЕРАТУРА

1. *Денисов В. И.* Математическое обеспечение системы ЭВМ-экспериментатор. М.: Наука, 1977.
2. *Дмитриенко М. Е., Шапиро А. М.* О способе организации пакета программ с динамической модульной структурой в среде ОС ЕС.— Программирование, 1983, № 6, с. 36—38.
3. *Стасышин В. М.* Диалоговая система оптимального планирования ДИСПЭК-1.— В кн.: Алгоритмическое и программное обеспечение задач оптимального планирования и проектирования. Новосибирск: НЭТИ, 1983, с. 49—57.

Поступила в редакцию 17.IX.1984

УДК 681.3.068

МЕТОД РАЗРАБОТКИ ПРОГРАММ И ПРОГРАММНЫХ СИСТЕМ (МЕТАПРОГРАММИРОВАНИЕ)

Шевель А. Е.

В данной работе рассматривается проблема разработки больших программ и программных систем. Предложен специальный язык для описания структуры программ. Описание структуры программы на этом языке называется метапрограммой. Метапрограмма определяет последовательность выполнения подпрограмм в программе.

Методы разработки программ и программных систем постоянно привлекают внимание разработчиков программного обеспечения. Важным моментом разработки является описание или представление структуры проектируемой программы.

Вопросы описания структуры программ неоднократно рассматривались в литературе. Так, теоретический аспект проблемы представлен, например, в [1, 2], где обсуждаются варианты языка логических схем и виды эквивалентностей алгоритмов. Показаны возможности формального

преобразования алгоритмов к более удобному виду, например к минимальному в некотором смысле.

Описание структуры программ затрагивается в известных технологиях по созданию программного обеспечения, например в [3, 4]. Создание и внедрение таких технологий были предприняты крупными фирмами-производителями программного обеспечения. Цель внедрения технологий — повышение надежности работы конечного программного продукта. Это достигается главным образом путем стандартизации проектной документации и рядом мер по организации работы исполнителей (например, метод главного программиста). В значительной степени такие технологии рассчитаны на внедрение в крупных коллективах разработчиков.

В [5] описывается язык проектирования программы, который применялся в основном для документирования проекта в период разработки и для обмена информацией между разработчиками. В литературе представлена еще одна сторона проблемы. Любая программа описывает последовательность действий над полем данных. Под действием можно понимать выполнение подпрограммы. В [6] в качестве программы рассматривается динамически формируемая цепочка действий, т. е. обращений к подпрограммам. Этот способ реализации программных систем применялся при разработке операционных систем и ряда прикладных систем [7—13].

Наибольший интерес представляет собой описание структуры программы в форме, допускающей простой ввод и интерпретацию на ЭВМ. Процесс интерпретации описания должен являться выполнением программы. Различные варианты способа описания программной структуры по передаче управления между подпрограммами рассматривались ранее в свете тех или иных приложений [14—18]. В [17, 18] такой способ был назван метапрограммированием.

Обсуждаемый ниже способ проектирования программ применяется в Ленинградском институте ядерной физики АН СССР. Метод с успехом использовался для создания программных систем на ЕС-1030 для обеспечения экспериментов лаборатории физики высоких энергий [10, 18]. Этот же способ разработки был применен при реализации диалогового текстового редактора [19]. В настоящее время он используется как для создания систем на ЕС-1030 и СМ-4, так и для подготовки программ, с помощью которых обрабатываются результаты конкретных физических экспериментов.

Цель настоящей работы — обсуждение метапрограммирования как способа разработки и описания больших программ. В частности, рассматривается конкретный вариант метаязыка для описания структуры программы по передаче управления от одной подпрограммы к другой.

МЕТАПРОГРАММИРОВАНИЕ

В период разработки, а также во время последующей эксплуатации любой большой программы часто возникает необходимость поменять порядок вызова подпрограмм, опустить обращение к некоторым подпрограммам или, напротив, добавить обращение к новым. Такое управление последовательностью обращений к подпрограммам можно реализовать с помощью массива параметров, который будет интерпретироваться, например, головной программой. Каждый элемент такого массива можно рассматривать как команду (инструкцию) интерпретатору на выполнение обращения к заданной подпрограмме или на изменение порядка интерпретации массива параметров. При этом команду интерпретатору умостно назвать метакомандой или метаоператором в отличие от команд или операторов, составляющих подпрограммы. Тогда последовательность метаоператоров естественно назвать метапрограммой, а такой способ организации и описание структуры программ — метапрограммированием.

Отметим некоторые следствия методичного применения предложенного подхода к построению программ:

метапрограмма является компактным документом по проектируемой программе;

способствует приданию разрабатываемой системе регулярной структуры;

метапрограмму можно составить в терминах той проблемы, решение которой должно быть получено на ЭВМ, т. е. метапрограмму может составить человек, незнакомый с техникой программирования на конкретной вычислительной машине;

метапрограмму, содержащую комментарии, можно сравнивать по выразительности с блок-схемой;

с использованием метапрограммы упрощается отладка программы, так как легко изменить последовательность выполнения программы путем изменения метапрограммы;

метапрограмма отражает реальную структуру программы.

Использование элементов метапрограммной организации программ — нередкое явление при разработке прикладных систем [6, 8, 12, 19]. В связи с этим полезно разработать и применять специальные средства для использования преимуществ метапрограммной организации.

Для решения этой задачи необходим конкретный язык описания структуры программ, а также транслятор и интерпретатор.

РЕАЛИЗАЦИЯ МЕТАЯЗЫКА

Операторы метаязыка служат для описания структуры программы по передаче управления от одной подпрограммы к другой. Метаоператоры не оказывают воздействия на работу подпрограмм, в связи с этим в метаязыке не рассматривается внутренняя структура подпрограмм и данных, с которыми они имеют дело. Тем не менее запись структуры программы на метаязыке позволяет охватить логику работы программы в целом. Такая запись полезна в дополнение к описанию работы подпрограмм и структур данных, а не вместо них.

Язык описания структуры программ включает в себя 11 операторов: *CHECK — оператор условного ветвления; *GOTO — оператор безусловного перехода; *BRANCH — оператор безусловного перехода с возвратом; *RETURN — оператор возврата; *NOP — пустой оператор; *EXIT — выход из метапрограммы; DEBUG — оператор включения прослеживания; *ENDEB — оператор выключения прослеживания; *PAGE — оператор управления листингом метапрограммы; т. е. переход на следующую страницу; XXX — оператор обращения к подпрограмме с именем «XXX»; *END — оператор конца метапрограммы.

Соглашения о кодировании достаточно тривиальны — один оператор занимает одну строку размером 80 байт. Оператор может располагаться в любом месте строки. Если в первой позиции слева находится символ* (звездочка), то это строка комментариев. Метки должны располагаться, начиная с первой слева позиции. Метапрограмма может начинаться любым оператором. Конец метапрограммы отмечается оператором *END.

Общий вид операторов в метапрограмме следующий:

[LABEL] OPERATOR [PARAMETER].

Рассмотрим выполнение некоторых операторов. Оператор *CHECK записывается следующим образом:

[LABEL] *CHECK N.

Здесь N — целое число больше нуля, обозначающее максимальное значение кода завершения подпрограммы. Этот оператор обеспечивает ветвление в зависимости от кода завершения. Если код завершения равен K, где

К меньше или равно N, то следующий после *CHECK будет интерпретироваться К-й оператор после *CHECK. Если К больше N, то при любом К будет выполняться N-й оператор после *CHECK.

Для отладки и прослеживания за ходом выполнения метапрограммы используются операторы *DEBUG и *ENDEB, которые предназначены для включения и выключения прослеживания соответственно

[LABEL] *DEBUG SUB

[LABEL] *ENDEB

Операнд SUB в операторе *DEBUG указывает на обращение к подпрограмме, которая, например, печатает общие области памяти или производит измерение времени выполнения различных подпрограмм. Если включен режим прослеживания, то перед интерпретацией каждой метаоперации будет выполнено обращение к подпрограмме, определенной в операторе *DEBUG.

Оператор обращения к подпрограмме имеет вид

[LABEL] XXX [/RES]

где XXX — имя подпрограммы; RES — имя ресурса.

При обращении к подпрограмме наличие в поле операнда имени ресурса означает следующее. Интерпретатор перед обращением к этой подпрограмме произведет захват ресурса и его освобождение после выполнения подпрограммы. Это свойство интерпретатора позволяет использовать одно тело программы в оперативной памяти для нескольких задач. Термины «задача» и «ресурс» имеют здесь то же значение, что и в операционных системах, например в ОС ЕС [20].

Любая подпрограмма может устанавливать значение кода завершения, который впоследствии можно проанализировать командой *CHECK. Параметры и данные от подпрограммы к подпрограмме передаются посредством общих областей памяти. Ответственность за интерпретацию параметров в подпрограммах лежит на разработчиках подпрограмм.

ТРАНСЛЯЦИЯ МЕТАПРОГРАММ

Трансляция метапрограмм выполняется транслятором, который написан на языке Фортран-IV. Процесс трансляции производится с использованием процедур языка управления заданиями ОС ЕС MGEN или MPT.

Процедура MGEN производит следующие действия:

транслирует метапрограмму и помещает результат трансляции (объектный код) в библиотеку;

готовит список адресов подпрограмм, имена которых используются в метапрограмме;

оформляет список адресов подпрограми отдельным программным модулем, который помещается в библиотеку загрузочных модулей.

Список адресов подпрограмм используется интерпретатором метапрограмм во время выполнения программы. Процедура MPT только транслирует исходную метапрограмму и помещает результат трансляции в библиотеку.

После выполнения MGEN разрабатываемую программу необходимо подготовить с помощью редактора связей. Во время сборки программы список адресов подпрограмм становится частью тела программы. Если метапрограмма конкретной программной системы готовится впервые, то по меньшей мере раз она должна быть оттранслирована процедурой MGEN.

После выполнения процедуры MPT достаточно запустить ранее подготовленную программу на исполнение. При этом будет использоваться список имен подпрограмм, построенной процедурой и включенный в тело программы во время сборки.

В связи с различием функций, возложенных на процедуры MGEN и MPT, различаются два типа метапрограмм: главные и дополнительные. Главная метапрограмма есть та, которая транслируется процедурой MGEN. Дополнительная метапрограмма транслируется с помощью процедуры MPT. Дополнительная метапрограмма может быть длиннее или короче главной. Однако если множество используемых имен подпрограмм в главной и дополнительной метапрограммах есть A и B соответственно, то B должно либо совпадать с A, либо быть подмножеством A.

Главной и дополнительной могут быть варианты одной метапрограммы. Например, после подготовки главной метапрограммы и сборки программы потребовалось удалить в целях отладки часть метапрограммы. Тогда полученный вариант будет дополнительным.

ИНТЕРПРЕТАЦИЯ МЕТАПРОГРАММ

Интерпретация метапрограммы выполняется подпрограммой — интерпретатором, которая интерпретирует метапрограмму в объектном виде. Подпрограмма-интерпретатор реализована в форме макроопределения ассемблера, что дает возможность генерировать различные варианты интерпретатора, отличающиеся рядом технических особенностей, например видом диагностики об ошибках.

Обращение к интерпретатору производится следующим образом:

CALL NAME (METPGM, IBLOK, SLIST),

где NAME — имя подпрограммы-интерпретатора; METPGM — массив целых слов, в который вводится метапрограмма в объектном виде перед обращением к интерпретатору; IBLOK — массив из четырех полуслов, представляющий собой управляющий блок интерпретатора, который должен находиться в общей области памяти; SLIST — адрес списка адресов подпрограмм.

В программе должен присутствовать хотя бы один оператор обращения к интерпретатору, например, в головной подпрограмме. Перед обращением к интерпретатору необходимо ввести оттранслированную метапрограмму из библиотеки в массив METPGM, что выполняется с помощью служебной подпрограммы MRDPGM. Кроме этого, нужно задать номер строки, с которой начнется интерпретация метапрограммы. Номер строки заносится в IBLOK(1).

Очевидно, что во время сборки проектируемой программы с помощью редактора связей интерпретатор и модуль списка адресов подпрограмм станут частью тела программы.

Любая подпрограмма системы может вырабатывать код завершения, значение которого может быть проверено оператором *CHECK. Подпрограмма заносит значение кода завершения в управляющий блок интерпретатора простым присваиванием IBLOK(4) = I, где I — значение кода завершения. Естественно, что управляющий блок должен быть доступен подпрограммам, устанавливающим код завершения.

Интерпретатор допускает рекурсивные обращения, поэтому любые подпрограммы системы могут обращаться к интерпретатору. Другими словами, свойство рекурсивности позволяет строить иерархическую структуру программы, используя на всех уровнях иерархии одну и ту же копию интерпретатора в оперативной памяти.

ЗАМЕЧАНИЯ ПО ИСПОЛЬЗОВАНИЮ МЕТАПРОГРАММ

С использованием рассматриваемой технологии параметры и данные передаются от подпрограммы к подпрограмме посредством общих областей памяти. Обращение к библиотечной подпрограмме можно выполнять посредством инструментальной подпрограммы, организующей обращение

к заданной подпрограмме. С другой стороны, в метапрограмме вовсе не обязаны фигурировать имена всех используемых подпрограмм.

Разработчик включает подпрограмму в качестве метаоперации или не включает ее, руководствуясь тем, что метапрограмма должна отражать программную структуру в целом, т. е. что и при каких условиях выполняется, а не как выполняется; метапрограмма должна быть детальной настолько, чтобы обеспечивались необходимые возможности по управлению программой путем модификации метапрограммы.

Использование метапрограмм способствует стройности структуры разрабатываемой программы, но не гарантирует ее. Метапрограммы эффективно использовать, когда вновь разрабатываемый программный текст составляет многие сотни или тысячи строк, которые представлены в виде десятков или сотен подпрограмм. Не менее эффективно применение метапрограммирования, когда используется много десятков или сотни ранее разработанных подпрограмм, к которым предполагается непосредственное обращение. В последнем случае новый программный текст может иметь более скромные размеры, чем указано выше.

В заключение заметим, что метапрограммный подход не является альтернативой какой-либо известной технологии подготовки программного обеспечения и может применяться с любой из них.

ЛИТЕРАТУРА

1. *Ершов А. П.* Введение в теоретическое программирование. М.: Наука, 1977.
2. *Криницкий Н. А., Миронов Г. А., Фролов Г. Д.* Программирование и алгоритмические языки. М.: Наука, 1979.
3. *Арефьева Н. А., Лушкина И. П., Родионов С. Т.* HIPO — метод разработки и сквозного документирования программ по принципу сверху вниз.— Управляющие системы и машины, 1978, № 3, с. 35—38.
4. *Вельбицкий И. В., Ходаковский В. Н., Шолмов Л. И.* Технологический комплекс производства программ на машинах ЕС ЭВМ и БЭСМ-6. М.: Статистика, 1980.
5. *Зелковиц М., Шоу А., Геннон Дж.* Принципы разработки программного обеспечения. М.: Мир, 1982.
6. *Голдингем Т.* Внутреннее функционирование системы 3.— В кн.: Супервизоры и операционные системы. М.: Мир, 1972, с. 89—91.
7. *Щепин В. С.* Макроинтерпретатор как основа языка моделирования.— Изв. АН СССР. Сер. Техническая кибернетика, 1972, № 3, с. 141—147.
8. *Brian C. D.* High speed array processing method applied to 2-D and 3-D image reconstruction.— IEEE Trans. Nucl. Sci., 1979, v. NS-26, № 2, p. 2718—2719.
9. *Lalive T.* Development of new real-time operating system «CHA-OS» at the Federal Institute of Technology. Transaction of the 1-st IFAC-IFIP symposium on software for computer control. Tallin, 1976.
10. *Сереброва Т. С., Суворова Л. Ф., Шевель А. Е., Яшкова С. В.* Диалоговое программное обеспечение физических экспериментов на синхроциклотроне ЛИЯФ АН СССР. Л.: препринт ЛИЯФ АН СССР, № 302, 1977.
11. *Куропаткин П. П.* Математическое обеспечение для настройки систем САМАС коллективного пользования. Л.: препринт ЛИЯФ АН СССР, № 519, 1979.
12. *Карлов А. А., Полинцев А. Д.* Локальное математическое обеспечение удаленной дисплейной станции. Дубна: препринт ОИЯИ, № 11—10967, 1977.
13. *Рябов Ю. Ф., Клопов Н. В., Новодворский Е. Г.* Программное обеспечение автоматизированной системы для накопления и экспресс-обработки экспериментальной информации на базе АСВТ М-4030. Л.: препринт ЛИЯФ АН СССР, № 418, 1978.
14. *Талыкин Э. А.* Модульное программирование в задачах сбора и обработки экспериментальных данных.— Автометрия, 1976, № 1, с. 65—72.
15. *Жук В. И., Шитиков Б. И.* Модульное программирование на разговорном языке описания блок-схем.— Автометрия, 1976, № 1, с. 79—83.
16. *Островной А. И.* Монитор для специализированных систем автоматизации экспериментов.— В сб.: Структура, технические средства и организация систем автоматизации научных исследований (Матер. 15-й Всесоюз. школы по автоматизации научных исследований). Л.: ЛИЯФ АН СССР, 1982, с. 106—116.
17. *Орешкин А. А., Сереброва Т. С., Шевель А. Е.* Метод разработки прикладных программ для обслуживания экспериментов по физике высоких энергий.— В сб.: Структура, технические средства и организация систем автоматизации научных исследований (Матер. 15-й Всесоюз. школы по автоматизации научных исследований.) Л.: ЛИЯФ АН СССР, 1982, с. 54—64.

18. *Орешкин А. А., Сереброва Т. С., Шевель А. Е.* Разработка прикладных диалоговых программ для обслуживания экспериментов по физике высоких энергий. Л.: препринт ЛИЯФ АН СССР, 1981, № 705.
19. *Грачева И. И., Орешкин А. А., Сереброва Т. С., Шевель А. Е.* Диалоговый редактор текста для ЭВМ средней производительности. Л.: препринт ЛИЯФ АН СССР, 1981, № 706.
20. *Наумов В. В., Пеледов Г. В., Тимофеев Ю. А., Чекалов А. Г.* Супервизор ОС ЕС ЭВМ. М.: Статистика, 1975, с. 66—79.

Поступила в редакцию 9.VII.1984
Переработанный вариант 10.VI.1985

УДК 681.3.068—181.4

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ УПРАВЛЕНИЯ НА ОДНОМ МИКРОПРОЦЕССОРЕ

Черемисинов Д. И.

Рассматривается задача разработки программы для микросистемы, используемой в качестве дискретного устройства управления. Описывается программная система, в которой в качестве средства представления алгоритмов управления используется язык Пралу, базирующийся на сети Петри.

Дискретные устройства, построенные на базе микропроцессорной техники, в зависимости от области их применения принято делить на а) микропроцессорные системы, предназначенные для замены традиционной вычислительной техники — микро-ЭВМ, и б) встроенные устройства, предназначенные для реализации одной определенной функции.

Системы типа б) часто называются контроллерами. Наиболее распространенным примером встроенной системы является устройство управления автоматической технологической машиной: станком, сборочной линией и т. д. Проектирование дискретного устройства на основе контроллера сводится к программированию последнего.

Применение контроллеров значительно упрощает изготовление дискретных устройств. Однако трудоемкость их проектирования остается того же порядка, что и при разработке на традиционной основе. Эти трудности в немалой степени обусловлены тем, что алгоритмы управления плохо представимы в традиционных понятиях программирования.

В [1—3] предложена модель дискретных устройств, на базе которой разработан простой, формальный и универсальный способ представления алгоритмов логического управления — язык Пралу [2]. В данной работе описывается набор операций, составляющих основу процедурного языка (ПЯ), позволяющего реализовать любой алгоритм на Пралу с использованием только одного процессора. Для демонстрации полноты предлагаемого набора операций рассматриваются конструкции языка Пралу и указываются эквивалентные выражения на ПЯ. Предлагаемый язык ПЯ использован при построении транслятора Пралу для микропроцессора К580. Транслятор позволяет автоматизировать преобразование алгоритмов логического управления в эффективные программы для контроллеров, построенных на базе этого микропроцессора. Язык Пралу кроме программирования микропроцессоров используется для проектирования электронных и релейных схем [4, 5], моделирования поведения дискретных устройств [6].

Для целей, поставленных в статье, удобно рассматривать язык Пралу как систему кодирования интерпретированных сетей Петри. Преобразование в программы похожей интерпретации сетей Петри рассматривалось в [7]. В описываемом трансляторе Пралу используется тот же принцип