Вестник Приднестровского университета. – Тирасполь, 1997. – № 2. – С. 74-79.

В С. Выхованец, О.Д. Выхованец

ПАРАЛЛЕЛЬНАЯ ПАРАДИГМА ПРОГРАММИРОВАНИЯ И ЕЕ ИСПОЛЬЗОВАНИЕ В ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

Существуют задачи, эффективное решение которых сопряжено со значительными трудностями при использовании последовательной модели вычислительного процесса и которые описываются множеством параллельно существующих и взаимодействующих объектов (процессов) различной природы. Эти задачи, как правило, имеют высокую трудоемкость и сложность программирования, что объясняется несколькими причинами

Во-первых, современные языки высокого уровня не имеют встроенных средств распараллеливания вычислительного процесса и поддерживают только последовательную модель вычислений.

Во-вторых, реализация параллельных процессов возможна в многозадачных операционных системах. При этом поддержка параллельных процессов сопряжена с достаточно высокими накладными расходами и не всегда эффективна для решаемой задачи.

В-третьих, такого рода задачи могут быть решены с использованием параллельных вычислительных систем, но эти системы дорогостоящи и малоэффективны для подавляющего большинства приложений, которые требуют сравнительно небольшого объема вычислений,

В-четвертых, проявляется психологическая инерция мышления разработчиков программных средств, заключающаяся в приверженности к устоявшимся традициям в программировании. Это приводит к усложнению постановки и реализации задачи при попытке ее сведения к известным схемам решений в рамках последовательной модели вычислений.

Таким образом, налицо явная методологическая проблема, которая заключается в противоречии между прикладной задачей и методами и средствами, используемыми для ее решения.

1. Парадигмы программирования

Впервые понятие парадигмы ввел в естествознание Т. Кун. Под парадигмой понимается исходная концептуальная схема или модель, принятая в качестве образца постановки проблем и их решения. Для конкретной области знания парадигма может быть построена как результат анализа и обобщения способа прогрессивного научного мышления, характерного для данной области. Причем выбор той или иной парадигмы для решения конкретной проблемы определяет методологическое обоснование методов ее решения и задает регулятивные основы мышления разработчика.

В области вычислительных наук можно выделить следующие парадигмы [1]: операционную, императивную, функциональную, структурную, объектно-ориентированную, последовательную, параллельную, потоковую, трансформационную, асинхронную, синхронную, дефинициональную, логическую, основанную на формах (бланках), демонстрационную, сетевую, парадигму капсулизации и локальную, введенную в [2].

Параллельная парадигма занимает особое место, поскольку выражает существующую сейчас тенденцию развития методов и средств параллельных вычислений [3]. Однако ее использование ограничивается только областью экспериментального и практического применения параллельных вычислительных систем.

Поставим целью расширить область использования параллельной парадигмы в программировании и разработать средства ее реализации в языках высокого уровня Для однопроцессорных ЭВМ и (или) однозадачных операционных систем.

2. Параллельное программирование

Под процессом будем понимать последовательность предусмотренных событий, которая определяется объектом или явлением и выполняется в заданных условиях. Под параллельным программированием будем понимать создание взаимодействующих процессов, выполнение которых совмещается во времени за счет использования различных ресурсов вычислительной системы.

Современные версии многозадачных операционных систем, поддерживающие параллельное программирование, перегружены множеством различных механизмов межпроцессного взаимодействия, что вызвано требованием обеспечения его полноты и попытками повышения его эффективности. Используемые механизмы плохо сочетаются друг с другом, имеют непрозрачную семантику и нерегулярную структуру, а это приводит к значительному усложнению прикладных программ.

Сведем многообразие существующих механизмов взаимодействия к взаимодействию процессов путем обмена сообщениями Передача-прием сообщения для участвующих во взаимодействии процессов является единым неделимым действием (примитивом), обладает функциональной полнотой, при этом служит только для обмена сообщениями между процессами и является средством синхронизации их функционирования, Под сообщением понимается адресуемая область памяти с заданным размером. Передача сообщения происходит путем пересылки сообщения процесса-отправителя в области памяти процесса-получателя

Анализ операционных систем показал, что любой процесс порождается, функционирует и уничтожается в неоднородной операционной среде, состоящей, как правило, из трех частей: самих параллельных процессов, разделяемых между процессами ресурсов (ресурсом может быть и обслуживание операционной системой) и асинхронно выполняемых процедур прерывания.

Для удовлетворения требования функциональной полноты взаимодействия процессов не только между собой, но и с операционной средой, использованы дополнительные механизмы взаимодействия процессов: семафорный механизм взаимного исключения для доступа к разделяемым ресурсам и асинхронный обмен сообщениями между процедурами прерывания и прикладными процессами

3. Особенности реализации

Параллельная модель вычислительного процесса реализована для языка программирования С в однозадачной операционной системе MS DOS с привязкой к архитектуре процессоров INTEL 80x86 Параллельным процессом является совокупность функции языка С и локальной области памяти - стека, выделенной для этой функции. В случае реентерабельности функции машинный код, составляющий ее тело, может быть использован параллельно несколькими процессами. Отметим, что в многозадачных операционных системах процессом является общесистемный объект (задача), не имеющий адекватного отражения в синтаксисе и семантике языков программирования высокого уровня.

Процесс может находиться в одном из следующих состояний: в состоянии выполнения EXEC, в состоянии готовности READY и в одном из четырех состояний ожидания: ожидания завершения процесса WAIT, ожидания передачи сообщения SEND, ожидания приема сообщения RECEIVE и ожидания транзакции TRANS Изменение состояния процесса происходит только при обращении к примитивам параллельного программирования (синхронное изменение состояния). Состояние SEND или RECEIVE прикладного процесса, ожидающего соответствующего взаимодействия с процедурой прерывания, может быть изменено в любой момент времени в результате обработки прерывания (асинхронное изменение состояния). Состояние ожидания транзакции TRANS соответствует ожиданию приема или передачи сообщения со стороны любого другого процесса. Каждый процесс находится в состоянии пассивного ожидания до наступления соответствующего события, после чего перехо-

дит в состояние готовности READY. Процесс прекращает свое существование, когда завершается выполнение соответствующей функции,

Каждый процесс имеет свой идентификатор - целое число, однозначно характеризующее этот процесс, и начальный и текущий приоритеты - целые числа в диапазоне от 0 до 255, Начальный приоритет определяет относительную частоту перехода процесса из одного состояния в другое и задается при порождении. Для исключения блокировки процесса в одном из своих состояний используется текущий приоритет Его значение устанавливается равным исходному при любом изменении состояния процесса и уменьшается при изменении состояния любого другого процесса, находящегося в том же начальном состоянии.

Для обеспечения концептуального единства реализации взаимодействие с внешней операционной средой сведено к стандартному взаимодействию прикладных процессов с помощью обмена сообщениями. Семафорный механизм взаимного исключения реализован путем создания процесса-семафора, отвечающего за взаимное исключение процессов при доступе к критическому ресурсу, а асинхронное взаимодействие прикладного процесса с процедурой прерывания сведено к использованию в процедуре прерывания двух дополнительных примитивов: асинхронного приема и асинхронной передачи сообщения, которые по своей семантике не отличаются от соответствующих примитивов взаимодействия прикладных процессов.

Примитивы, необходимые для поддержки параллельной парадигмы программирования, разбиты на несколько функциональных групп: управление процессами - создание процесса creat и приостановка процесса suspend; прием-передача сообщения - послать сообщение send и принять сообщение receive; образование барьера - ожидание завершения процесса wait и ожидание транзакции trans; проверка состояния - получить состояние процесса state; асинхронный обмен сообщениями - послать сообщение прикладному процессу asend и принять сообщение от прикладного процесса areceive.

Использование синхронного изменения состояния процессов, пассивного ожидания и динамически изменяемого текущего приоритета позволило уменьшить затраты на поддержку процессов с различным временем существования и различной интенсивностью переключения. Вычислительный эксперимент, проведенный на ПЭВМ с процессором INTEL 80286 и тактовой частотой 12 МГц, показал, что время переключения процессов составляет не более 90 мкс.

Заголовочный файл модуля поддержки параллельной парадигмы программирования и исходный текст реентерабельной функции для процесса-семафора приведен в Приложении.

4. Применение

Опыт использования параллельной парадигмы программирования в описанной реализации позволил сократить время разработки и отладки программных средств, повысить их надежность, уменьшить расходы на сопровождение. Так, например, создание программного обеспечения для однокристальной микроЭВМ источника бесперебойного питания выявило значительные трудности программирования в рамках последовательной модели без создания специальных языковых выразительных средств. Их преодоление стало возможно только после пересмотра принципов реализации в пользу параллельной модели вычислительного процесса и

использования параллельной парадигмы программирования. Удалось также добиться сокращения времени программирования и отладки задач моделирования сетевых протоколов различного назначения при курсовом и дипломном проектировании

Список задач эффективного применения параллельной парадигмы программирования в описанной реализации можно легко продолжить. Сюда можно отнести задачи автоматизации научных исследований, управления сложными техническими системами в реальном масштабе времени, диагностирования и отладки аппаратуры и приборных комплексов, а также задачи создания компактных контроллеров различного рода приборов, устройств и технологического оборудования.

Приложение

П.1.. Заголовочный файл proc.h

```
typedef enum {PROC_NULL=0,PROC_MAIN, PROC_INT} procjd;
typedef enum{UNDEF, STOP, EXEC, READY, WAIT, SEND, RECEIVE, TRANS} proc_st;
typedef unsiqned char proc_prt;     typedef unsiqned int data_len;
typedef void *data_ptr;             typedef void     (*func_ptr)();
procjd creat (func_ptr, proc_prt),  /*Создание */
proc_st state (proc_id),            /* Проверить состояние  */
void wait (proc_id);                /* Ожидание завершения */
void suspend (proc_prt);            /* Приостановка */
proc_id trans (proc_state);         /* Ожидание транзакции  */
data_len send (procjd, data_ptr, data_len);      /* Передача  */
data_len asend (proc_id, data_ptr, data_len);  /* Ас. передача */
data_len receive (proc_id, data_ptr, data_len); /* Прием */
data_len areceive (proc_id, data_ptr, data_len); /* Ас. прием */
```

П2. Процесс-семафор

```
#include "proc. h"
void semafor (proc_id self, proc_id parent, proc_prt prt) {
    int sem = 0, done = 0; proc_st st; procjd p;
    receive (parent, sem, sizeof sem);     /* Прием параметров */
    while (!done) { /*Цикл, пока parent не уничтожит семафор */
          p = trans( (sem 0) ? RECEIVE | SEND : SEND); st = state (p);
          if( st == RECEIVE) {sem--; send(p,   sem, sizeof sem);}
          else if(st == SEND) {
                    receive( p,   done, sizeof done), sem++;
                    if( p!= parent) done =0;}
    }
    return;
}
```

Цитированная литература

1. Ambler A.L., Burnett *MM.,* Zimberman B.A. Operation versus definitional: A perspectiv on programming paradigms // Computer. 1992. Vol. 25, № 9. C 28-43.

2 Выхованец В.С., Гордиенко К.П. Процессор с сокращенным набором команд // Вестник Приднестровского университета. 1995. № 1. С. 124-128.

3. Миренков Н.Н. Параллельное программирование для многомодульных вычислительных систем. М.: Радио и связь, 1989.