Gem #81: Семафоры GNAT.

Автор: Pat Rogers , AdaCore

Краткое содержание: Gem Ada #81 - В ранее опубликованном Gem (#70, Идиома Scope Lock) обсуждалась иногда возникающая необходимость применять механизм низкоуровневой синхронизации для защищённой объектной конструкции высшего уровня. Приведенный код ссылался на возможности пакета семафоров в иерархии GNAT. В Gem Ada #81 будут исследованы абстрактные представления, предоставляемые этим пакетов, в частности - проектные решения.

Давайте начнём…

Существует много определений семафорных операций, несмотря на то, что общее понятие остаётся, как представлено Dijkstra в 1968. Мы предполагаем, что у читателя есть некоторое знакомство с семафорами и их семантикой, и таким образом, мы не будем их здесь освещать. Много книг доступны для тех, которые запрашивают дополнительную информацию. кому требуется дополнительная информация. См., в частности, «Принципы параллельного и распределённого программирования М. Бен-Ари». (Principles of Concurrent and Distributed Programming – M. Ben-Ari).

Пакет GNAT.Semaphores определяет две семафорных абстракции, обоих с точки зрения защищённых типов. Мы используем защищённые типы, вместо того, чтобы определить абстрактные типы данных на основе средств операционной системы, ради мобильности. Простая обёртка вокруг средств операционной системы или RTOS была бы разумна, но получающемуся интерфейсу (и реализация, конечно) не обеспечит требуемую мобильность иерархии GNAT общего назначения.

Первая абстракция определяет семафор "подсчета", в котором число целых чисел поддерживается операциями, так что операция получения выполняется (и, таким образом, возвращается, позволяя вызывающему объекту продолжить), когда число больше нуля. Счётные семафоры удобны для выражения условий синхронизации с точки зрения наличия некоторого количества ресурсов. Например, с круговым ограниченным буфером количество семафоров может представлять количество доступных слотов буфера. Вызывающие объекты должны ждать, пока буфер не будет заполнен при попытке вставить новый элемент, и блокирующий вызов для получения семафора достигнет этого эффекта напрямую.

Декларация для счётного семафора выглядит следующим образом. Обратите внимание, что мы удалили комментарии в строке для краткости, но рассмотрим их содержимое.

```
protected type Counting_Semaphore
   (Initial_Value : Natural;
   Ceiling : System.Priority)
is
   pragma Priority (Ceiling);

   entry Seize;
   procedure Release;

private
   Count : Natural := Initial_Value;
end Counting_Semaphore;
```

Второй Тип реализует "бинарные" семафоры. Этот вид семафора является менее гибким, чем «счётный» семафор, в том, что понятие счётчика не поддерживается. Абстракция «бинарный семафор» – это просто флаг, указывающий, доступен ли семафор. (Если значение счётного семафора варьируется от нуля до единицы, эффект будет одинаковым с «бинарным семафором».)

```
protected type Binary_Semaphore
```

```
   (Initially_Available : Boolean;
    Ceiling : System.Priority)
is
   pragma Priority (Ceiling);

   entry Seize;
   procedure Release;

private
   Available : Boolean := Initially_Available;
end Binary_Semaphore;
```

В обоих случаях предоставляемые операции состоят из записи "`Seize`" для получения семафора с взаимоисключающим доступом и защищённой процедуры "`Release`" для его освобождения. Операция захвата (`Seize`) должна быть записью для барьера, выражающей требуемую синхронизацию условий. Выпуск (`Releasing`) семафора не имеет такого требования, поэтому он не должен быть записью.

Оба типа явно определены как защищённые типы, чтобы пользователи могли выполнять условные и временные вызовы, когда это необходимо. Эта возможность устраняет одну из проблем переносимости семафоров. Хотя основные операции "`acquire`" (получение) и "`release`" (выпуск) всегда будут предоставляться (под любым именем), для других форм взаимодействия не существует "стандартного" интерфейса. Определённые языком конструкции обеспечивают некоторые из этих видов взаимодействий и делают реализацию переносимой.

Оба типа требуют дискриминантов. Тип счётного семафора использует дискриминант для указания начального неотрицательного значения счётчика. Тип двоичного семафора использует логический `Boolean` дискриминант, чтобы указать, должен ли семафор быть изначально доступен. Например, двоичный семафор будет изначально доступен при использовании для выражения взаимного исключения, но может семафор быть изначально недоступен при использовании для выражения условий синхронизации.

Кроме того, оба типа используют другой дискриминант для указания максимального приоритета для объектов типа. Затем дискриминант передаётся в качестве аргумента в `Pragma Priority`. Если Real-Time Systems Annex (Приложение о системах реального времени) действует, то эта часть имеет существенно важное значение; в противном случае она не имеет никакого эффекта. Обратите внимание, что значение по умолчанию не может быть предоставлено для дискриминанта `Ceiling`, потому что это потребует значения по умолчанию и для другого дискриминанта. Лучшее, что мы можем сделать, это определить удобное значение, которое будет использоваться в объявлениях отдельных объектов, когда Real-Time Annex (Приложение реального времени) не действует, поэтому в `GNAT.Semaphores` предусмотрена следующая константа:

```
Default_Ceiling : constant System.Priority := System.Default_Priority;
```

Однако подтипы могут использоваться для повышения удобства, удобочитаемости и надежности. Более ранний Gem (# 70) привёл пример:

```
subtype Mutual_Exclusion is Binary_Semaphore
   (Initially_Available => True,
    Ceiling             => Default_Ceiling);
```

Подтип гарантирует, что соответствующие объекты изначально доступны, но также удобно предоставляет значение дискриминанта `Ceiling` (при условии, что Real-Time Annex "приложение в реальном времени" не действует) и даёт коду выполняющий чтение индикацию на предполагаемое использование.

Как видите, объявления защищённых типов могут быть очень выразительными. Дискриминанты – это та часть, которую вы, возможно, не рассматривали, хотя это отнюдь не необычный подход. Защищённые тела обоих типов тривиальны и не будут показаны здесь. Вы можете видеть их в иерархии GNAT вместе с реализациями всех других пакетов иерархии GNAT.

Связанный со статьёй текст программы

Attached Files отсутствуют

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Пэт Роджерс (Pat Rogers) был профессионалом в области вычислительной техники с 1975 года, в основном работая над приложениями на основе микропроцессора в режиме реального времени на языках Ada, C, C ++ и других языках, включая высокопроизводительные имитаторы полёта и системы контроля и сбора данных (SCADA), контролирующие опасные материалы. Впервые узнав Аду в 1980 году, он был директором лаборатории Ada9X для совместной программы Advanced Strike Technology для ВВС США, исследователя принципов в распределённых системах и исследовательских проектов отказоустойчивости с использованием Ada для ВВС США и армии, а также помощника директора по исследованиям в NASA Software Engineering Research Center. У него есть B.S. и M.S. степени в области проектирования компьютерных систем и компьютерных наук Университета Хьюстона и доктора философии. в информатике из Университета Йорка, Англия. Являясь членом старшего технического персонала AdaCore, он специализируется на поддержке разработчиков в режиме реального времени / встроенных систем, создаёт и предоставляет учебные курсы, а также является руководителем проекта и разработчиком плагина GNATbench Eclipse для Ada. Он также имеет черный пояс 3-го Дан в Тэ Квон До и является основателем клуба AdaCore «The Wicked Uncles»..

Last Updated: 10/13/2017
Posted on: 3/8/2010

Обсуждение…