Gem #35: Пакет кольцевых буферов в иерархии GNAT (Часть 1)

Автор: Pat Rogers, AdaCore

Краткое содержание: Gem Ada #35 - В Ada95 были введены так называемые «защищённые типы», представляющие собой фундаментальные элементы, необходимые для эффективного многопоточного программирования и обработки прерываний. В данном Gem описывается использование защищённых типов в ходе реализации классических асинхронных буферных абстракций, предоставляемых иерархией элементов GNAT. Предполагается, что по крайней мере в какой-то степени Вам знакомо понятие языка Ada «защищённые типы», поэтому их семантика объясняется лишь частично.

Давайте начнем…

Ограниченный буфер представляет собой классический компонент параллельного программирования, демонстрирующий асинхронные взаимодействия задач. Концепция - это буфер фиксированного размера, к которому обращаются несколько задач, некоторые задачи выполняют вставки элементов и некоторые задачи выполняют их удаление одновременно и асинхронно. Следовательно, реализация буфера должна быть защищена от «условий гонки», при которых задачи обращаются к реализации в произвольном чередовании и тем самым могут повреждать содержимое буфера. В дополнение к этому «взаимоисключающему доступу» буфер также требует «синхронизации условий», в котором вызывающие абоненты находятся в ожидании, пока запрашиваемый буфер не будет иметь необходимое состояние. Например, задача не может удалить элемент из буфера, когда буфер пуст. Аналогично, элемент не может быть помещён в буфер, когда буфер заполнен.

До Ada 95 программисты, желающие писать переносимый код, должны были использовать рандеву (механизм языка программирования Ada) для достижения взаимного исключения, с защитой для реализации синхронизации условий, поскольку язык не предоставлял никакой другой механизм синхронизации. Хотя расширенное рандеву имеет ряд преимуществ и является шагом вперёд в разработке языка, оно имеет значительные накладные расходы по сравнению с механизмами более низкого уровня, такими как семафоры, и также является синхронным механизмом. (Ada 80 имел встроенный тип задачи «Семафор», предназначенный для эффективного использования, но смешивание рандеву более высокого уровня с абстракцией семафора более низкого уровня считалось плохим стилем программы на языке Ada.) Кроме того, рандеву доступно только между задачами, что означает, что буфер должен быть реализован как задача, например, как потоки доступа. В результате вставка и удаление элементов потребует дорогостоящего переключения задач, что является основным источником сравнительной неэффективности.

Конструкция защищённого типа, добавленная в Ada 95, напрямую обращается к этой проблеме. Защищённые типы обеспечивают эффективный взаимоисключающий доступ к инкапсулированным данным с прямым выражением синхронизации условий, когда это необходимо. Защищённые типы не определяют потоки управления, поэтому их использование не связано с переключением задач, и хотя они делают больше, чем простые семафоры, их накладные расходы сопоставимы.

Иерархия пакетов GNAT включает в себя общий пакет GNAT.Bounded_Buffers, предоставляющий только ту абстракцию, которую мы имеем в виду, параметризуемую для общего использования. Реализация ограниченного буфера будет представлять собой массив, и мы будем выполнять присвоения значений, хранящихся в любом заданном буфере, поэтому общий формальный тип, представляющий значения, объявляется частным, но не ограниченным частным или неопределённым:

```
generic
   type Element is private;
package GNAT.Bounded_Buffers is
```

Учитывая этот общий формальный профиль, пользователи могут создавать типичные данные по мере необходимости. Например, учитывая соответствующий общий тип фактического параметра с именем «Job», мы можем создать его следующим образом:

```
package Jobs is new GNAT.Bounded_Buffers (Element => Job);
```

Объявление пакета содержит pragma Pure, это универсальный подход, который можно использовать во время разработки библиотечного блока без потенциальной проблемы доступа до разработки. Этот эффект достигается благодаря тому, что в дополнение к другой семантике независимые единицы предварительно прорабатываются.

Затем пакет объявляет Тип массива, используемый внутри в представлении типа ограниченного буфера:

```
type Content is array (Positive range <>) of Element;
```

Тип массива должен быть объявлен вне защищённого типа, а не внутри в частной части как скрытый артефакт реализации. Это неудачное ограничение из-за того, что защищённые типы изначально назывались «защищёнными записями», с семантикой типа записи: типы записей не могут объявлять такие вещи, как другие типы! Это ограничение было известно во время пересмотра Ada 2005, но другие аспекты пересмотра были более важными, поэтому это нежелательное ограничение остаётся.

Следующее объявление в пакете является постоянным значением типа System.Priority:

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

В приложении реального времени, использующем приложение «Системы реального времени»(the Real-Time Systems Annex), защищённым типам присваивается «потолочный» приоритет (a “ceiling” priority). Объявленная здесь константа является дефолтом для этой цели, так что программы, не использующие Real-Time Systems Annex, могут игнорировать этот аспект.

Наконец, пакет объявляет сам защищённый тип с двумя дискриминаторами:

```
protected type Bounded_Buffer
   (Capacity : Positive;
    Ceiling  : System.Priority)
is
   pragma Priority (Ceiling);
```

Первый дискриминант - это ёмкость объекта экземпляра, то есть максимальное количество значений, которое оно может содержать. Это значение будет использоваться в объявлении объекта скрытого массива типа Content. При таком подходе разные объекты одного типа буфера могут иметь разные возможности. Второй дискриминант представляет значение приоритета потолка, используемое в приоритете pragma. Здесь константа Default_Ceiling будет использоваться в приложениях, отличных от реального времени. Обратите внимание: мы не можем использовать константу Default_Ceiling как значение дискриминанта по умолчанию, потому что язык не позволяет некоторым дискриминаторам иметь значения по умолчанию, если у всех нет значений по умолчанию.

Продолжая работу с примером экземпляра «Jobs», объявление ограниченного буфера указывает эти значения дискриминанта:

```
Buffer : Jobs.Bounded_Buffer (Capacity => 20,
                              Ceiling => Jobs.Default_Ceiling);
```

В этом примере мы произвольно установили Ёмкость буфера на 20. Обратите внимание, что Тип Bounded_Buffer предоставляется непосредственно как защищённый Тип, а не как ограниченный

закрытый Тип, завершённый защищённым типом. При таком подходе клиенты имеют полную гибкость для выполнения всех защищённых типов, таких как временные и условные вызовы.

Далее защищённый Тип объявляет видимые операции. Две основные операции-Insert и Remove, определяемые как записи для барьеров, которые определяют требуемую синхронизацию условий. (Только защищённые записи могут иметь барьеры, в отличие от защищённых процедур и функций.) Барьеры выражают ”не полные“ и ”не пустые" условия и заставляют своих абонентов ждать, пока эти условия наступят.

```
entry Insert (Item : Element);
entry Remove (Item : out Element);
```

Объявляются три функции. Имена `Empty` ("Пустой") и `Full` (”Полный") описывают назначение первых двух функций. Третья функция, `Extent` ("Экстент"), возвращает количество элементов, содержащихся в буфере. Стоит отметить, что состояние буфера, к которому могут быть применены эти функции, может измениться сразу после возврата вызова.

```
function Empty return Boolean;
function Full return Boolean;
function Extent return Natural;
```

Во второй части этого Gem (Gem #37) мы рассмотрим приватную часть защищённого типа, тело пакета и тело защищённого типа.

Связанный исходный код

Attached Files

```
with GNAT.Bounded_Buffers;

with Ada.Text_IO;    use Ada.Text_IO;

procedure Demo_Buffers is

   type Job is new Integer;  --  for example

   Stop : constant Job := -1;  --  arbitrary

   package Jobs is new GNAT.Bounded_Buffers (Job);

   Buffer_Size : constant := 20;  --  arbitrary
   Iterations  : constant := Buffer_Size * 3;  --  arbitrary
      --  we want to insert more items than the buffer can
      --  hold at one time to ensure that the condition
      --  synchonization is exercised

   Buffer : Jobs.Bounded_Buffer (Capacity => Buffer_Size,
                                 Ceiling  => Jobs.Default_Ceiling);

   task Producer;

   task Consumer;


   task body Producer is
   begin
      for K in Job range 1 .. Iterations loop
         --  do some work to produce the next value;
         --  in this case we just insert the loop parameter
         --  ...
         Buffer.Insert (K);
         --  ...
         --  do some more work
```

```
         --  ...
      end loop;

      Buffer.Insert (Stop);
   end Producer;


   task body Consumer is
      Next : Job;
   begin
      loop
         Buffer.Remove (Next);
         exit when Next = Stop;
         --  process the next job ...
         Put_Line (Next'Img);  --  for example
         Flush;
      end loop;
   end Consumer;

begin
   null;
end Demo_Buffers;
```

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

Об авторе

Пэт Роджерс 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 полета и системы контроля и сбора данных (SCADA), контролирующие опасные материалы. Впервые узнав язык программирования Ada 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 программ, которые работают в режиме реального времени и для 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: 5/12/2008

Обсуждение…