Gem #13: Идиомы для работы с прерываниями (Часть 1)

Автор: : Pat Rogers, AdaCore

Краткое содержание: Gem Ada #13 - Для работы с прерываниями в языке Ада обычно используются две идиомы. Характеристики одной из них более подходят под определение качественной разработки программного обеспечения, в то время как у другой лучшие рабочие показатели. В данном gem рассматриваются эти две идиомы.

Давайте начнем…

Напомним, что в Ada защищенные процедуры являются стандартным механизмом обработки прерываний. Такой подход имеет ряд преимуществ перед «традиционным» использованием незащищенных процедур. Во-первых, обычные процедуры не имеют приоритета, но защищенные объекты могут иметь приоритет прерывания и таким образом интегрированы с общей семантикой приоритета. Выполнение записей и процедур в защищенном объекте будет выполняться на этом уровне, и только прерывания более высокого уровня могут вытеснить это выполнение. Таким образом, условия гонки не возможны. Кроме того, синхронизация состояния выражается непосредственно через входные барьеры, что позволяет легко выразить и понять взаимодействие с другими частями системы. Наконец, защищенные объекты поддерживают локализацию данных и их манипуляционные процедуры, а также локализацию нескольких обработчиков прерываний в пределах одного защищенного объекта, когда каждый из них нуждается в доступе к локальным данным.

Ответ на прерывания часто выполняется на уровнях, причем обработчик первого уровня обеспечивает очень быстрый отклик, который делает ограниченную обработку, и обработчик вторичного уровня, который выполняет более дорогостоящую обработку за пределами контекста прерывания, на приоритет уровня приложения. Естественным выражением этой структуры является использование защищенной процедуры как обработчика первого уровня и задачи как вторичного уровня. Защищенная процедура реагирует на прерывание, а затем сигнализирует задачу, когда она должна работать.

Например, рассмотрите обработку сообщений через UART (универсальный асинхронный приемник-передатчик), в котором прерывание сигнализирует о поступлении первого символа. Процедура обработчика прерываний могла бы захватить этот символ, поместить его в буфер внутри защищенного объекта, а затем либо опросить остальные символы (если необходимо), либо сбросить для следующего прерывания. После получения всего сообщения защищенная процедура может затем сигнализировать задачу вторичного обработчика для синтаксического анализа сообщения и соответствующего ответа.

Мы будем использовать такой процессор сообщений, используя идиомы дизайна. В каждом случае мы инкапсулируем оба уровня кода обработки прерываний внутри тела пакета с именем Message_Processor. Вся обработка выполняется в теле пакета, и ничего не экспортируется, что требует завершения. Следовательно, в декларации нужна прагма Elaborate_Body, чтобы сделать тело пакета законным. Содержимое объявления пакета, как показано ниже, вероятно, пойдет и в теле, но для целей этого GEM мы оставим их здесь.

```
package Message_Processor is

  pragma Elaborate_Body;
```

```
  subtype Message_Size is Integer range 1 .. 256; -- arbitrary

  type Contents is array (Message_Size range <>) of Character;

  type Message (Size : Message_Size) is
    record
      Value  : Contents (1..Size);
      Length : Natural := 0;
    end record;

end Message_Processor;
```

Первая идиома проекта

В первой идиоме защищенный обработчик процедуры первого уровня сигнализирует обработчику задачи второго уровня, и включает барьер на защиту записи в том же защищенном объекте. Задача второго уровня приостанавливается на вызове записи и, когда позволено возобновляет выполнение, произвадит вторичную обработку. Параметры записи могут использоваться, чтобы передать информацию к задаче, например сообщение, полученное на UART. Код для этой идиомы приводит к тому, что тело пакета структурировано следующим образом: Код для этой идиомы приводит к телу пакета, структурированному следующим образом:

```
with UART;
with System;
with Ada.Interrupts;

package body Message_Processor is

   Port : UART.Device;

   protected Receiver is ... -- the first-level handler

   protected body Receiver is ...

   task Process_Messages is ...

   task body Process_Messages is ...

begin
   UART.Configure (Port, UART_Data_Arrival, UART_Priority);
   UART.Enable_Interrupts (Port);
end Message_Processor;
```

В вышеупомянутом защищенный объектный `Receiver` - обработчик первого уровня; вторичный обработчик - задача `Process_Messages`. Аппаратные средства UART представлены объектом под названием типа `Port`, определенного UART пакета (не показанный). Исполнимая часть тела пакета автоматически конфигурирует UART и включает его прерывания после того, как защищенный объект и задача будут созданы.

Защищенный объект Receiver содержит процедуру обработки прерываний; запись, вызываемую задачей вторичного обработчика; буфер, содержащий текущие символы; и логическую переменную, используемую для входного барьера:

```
   UART_Priority     : constant System.Interrupt_Priority   := ...
   UART_Data_Arrival : constant Ada.Interrupts.Interrupt_Id := ...

   protected Receiver is
      entry Wait (Msg : access Message);
      pragma Interrupt_Priority (UART_Priority);
```

```
private
   procedure Handle_Incoming_Data;
   pragma Attach_Handler (Handle_Incoming_Data, UART_Data_Arrival);
   Buffer        : Contents (Message_Size);
   Length        : Natural := 1;
   Message_Ready : Boolean := False;
end Receiver;
```

Pragma Interrupt_Priority присваивает данный приоритет всему защищенному объекту. Никакие другие прерывания на уровне или ниже этого уровня не будут включены во время выполнения процедуры. Обратите внимание, что процедура объявлена в частной части защищенного объекта. Размещение там исключает «случайные» вызовы от клиентского программного обеспечения в будущих мероприятиях по техническому обслуживанию. Обратите внимание также на прагму Attach_Handler, которая постоянно связывает процедуру с прерыванием.

В теле охраняемого объекта у нас есть тела для входа и процедуры. Запись управляется логическим Message_Ready, для которого установлено значение True, когда обработчик прерываний определяет, что все символы были получены для данного сообщения. Тело ввода копирует содержимое буфера непосредственно в объект сообщения вызывающего объекта и затем сбрасывает буфер для следующего прихода сообщения.

```
protected body Receiver is

   entry Wait (Msg : access Message) when Message_Ready is
   begin
      Msg.Value (1 .. Length) := Buffer (1 .. Length);
      Msg.Length := Length;
      -- reset for next arrival
      Length := 1;
      Message_Ready := False;
   end Wait;

   procedure Handle_Incoming_Data is
   begin
      UART.Disable_Interrupts (Port);
      Buffer (1) := UART.Data (Port);
      -- poll for all remaining
      while UART.Data_Available (Port) loop
         Length := Length + 1;
         Buffer (Length) := UART.Data (Port);
      end loop;
      UART.Enable_Interrupts (Port);
      -- wake up the task
      Message_Ready := True;
   end Handle_Incoming_Data;

end Receiver;
```

В этом примере процедура обработчика прерываний использует подход опроса. Сначала он отключает дальнейшие прерывания от UART, а затем захватывает все входящие символы. Наконец, при повторном включении устройство прерывает и включает запись, настроив Message Ready на True.

У задачи обработчика второго уровня нет собственных записей, потому что ничто не вызывает ее. Мы только должны установить приоритет задачи, как определено в Конфигурации пакета (не показанный Config.Process_Messages_Priority), который определяет все приоритеты приложения.

```
task Process_Messages is
   pragma Priority (Config.Process_Messages_Priority);
end Process_Messages;

task body Process_Messages is
   Next_Message : aliased Message (Size => Message_Size'Last);
begin
   -- any initialization code
   loop
      Receiver.Wait (Next_Message'Access);
      -- process Next_Message ...
   end loop;
end Process_Messages;
```

Задача приостанавливается до тех пор, пока запись не будет выполнена, а затем обработает сообщение определенным образом.

На следующей неделе мы рассмотрим вторую идиому дизайна, а затем сравним их. Оставайтесь с нами ...

Обсуждение…