Gem #14: Идиомы для работы с прерываниями (Часть 2)

Автор: : Pat Rogers, AdaCore

Краткое содержание: Gem Ada #14 - 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…

На прошлой неделе мы представляли эту тему и исследовали первый из двух проектов. Суть его - то, что мы хотим реализовать многоуровневый ответ на прерывания, в которых защищенная процедура реализует обработчик первого уровня, и задача реализует вторичную обработку уровня за пределами контекста прерывания. Мы используем последовательную передачу данных по UART (Универсальный Асинхронный Передатчик Получателя) как пример. В обоих проектах мы инкапсулируем два уровня кода обработки прерываний в теле пакета по имени Message_Processor.

Вторая идиома проекта

Вторая идиома аналогична первой, поскольку защищенный объект инкапсулирует процедуру обработки прерываний, но эта вторая конструкция использует объект Suspension_Object вместо защищенной записи для сигнализации задачи. Тип Suspension_Object объявляется в пакете Ada.Synchronous_Task_Control и по существу является булевым флагом с синхронизацией. Одна задача может ожидать, что данный объект Suspension_Object станет «истинным» и будет приостановлен до тех пор, пока не будет установлено другое задание. Полученная структура для корпуса пакета выглядит следующим образом:

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

package body Message_Processor is

  Port : UART.Device;

  package STC renames Ada.Synchronous_Task_Control;

  Message_Ready : STC.Suspension_Object;

  Buffer : Contents (Message_Size);
  Length : Natural := 1;

  protected Receiver is ...

  protected body Receiver is ...

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

Существенные различия в структуре включают новый объект Suspension_Object и буфер, содержащий последнее полученное сообщение, перемещенное из защищенного объекта в декларативную часть тела пакета.

Защищенный объект Receiver больше не объявляет запись, поскольку использует Message_Ready для сигнализации задачи. Локальные данные также не требуются. Требуется только процедура обработки прерываний.

```
protected Receiver is
  pragma Interrupt_Priority (UART_Priority);
private
  procedure Handle_Incoming_Data;
  pragma Attach_Handler (Handle_Incoming_Data, UART_Data_Arrival);
end Receiver;
```

Тело процедуры идентично той из другой идиомы проекта за исключением того, что это устанавливает Suspension_Object в Истину вместо того, чтобы установить локальную логическую переменную, используемую в барьере записи.

```
protected body Receiver is

  procedure Handle_Incoming_Data is
  begin
    -- as before
    ...

    STC.Set_True (Message_Ready);
  end Handle_Incoming_Data;

end Receiver;
```

Теперь задача делает работу, которую запись сделала в другом проекте. Это сначала ожидает Message_Ready, чтобы быть Правда, приостанавливая при необходимости. Тогда это копирует буферное содержание в локальный объект сообщения и сбросы для следующего прибытия. Обратите внимание на то, что, когда Suspend_Until_True выполняется, указанный Suspension_Object автоматически установлен в Ложь на выходе.

```
task body Process_Messages is
  Next_Message : aliased Message (Size => Message_Size'Last);
begin
  -- any initialization code here...
  loop
    STC.Suspend_Until_True (Message_Ready);
    -- capture the new message in Buffer
    Next_Message.Value (1..Length) := Buffer (1..Length);
    Next_Message.Length  := Length;
    -- reset for next arrival
    Length := 1;
    -- process Next_Message ...
  end loop;
end Process_Messages;
```

Сравнение идиом проектов

Первая идиома, в которой запись используется, чтобы сигнализировать обработчик второго уровня, является лучшим проектом с точки зрения разработки программного обеспечения: функциональность сгруппирована с данными, которыми управляют (буфер локален для защищенного объекта), синхронизация условия непосредственно выражена через барьер записи и не реализована задачей вызова, и коммуникация выполнена через

параметры записи. Все эти характеристики приводят к более удобной в сопровождении, устойчивой реализации.

Однако семантика защищенных объектов подразумевает затраты времени выполнения, которые, хотя и относительно небольшие, больше, чем у объекта Suspension_Object. Вторая идиома, скорее всего, будет быстрее первой, предполагая достойную реализацию типа Suspension_Object. Однако размещение всего кода в корпусе пакета ограничивает вредные эффекты результирующей структуры.

Средства обработки и управления прерываниями, предоставляемые Ada, определены в Systems Programming Annex, раздел C.3 Reference Manual. Поддержка обширна и заслуживает изучения.

Обсуждение…