Gem #77: Куда делась моя память? (Часть 1)

Автор: Emmanuel Briot, AdaCore

Краткое содержание: Gem Ada #77 - Для контроля за использованием памяти, обнаружением утечек памяти и разрешения проблем общего характера, связанных с памятью, существует множество инструментов и библиотек. В данной и нескольких последующих публикациях будет предложен обзор этой тематики и описание работы с подобными инструментами.

Давайте начнём…

Если ваш стандарт кодирования не запрещает динамическое выделение, управление памятью является постоянной проблемой при разработке системы. Может потребоваться ограничить объем памяти, необходимый приложению, или могут возникнуть утечки памяти (блоки выделения, которые никогда не возвращаются в систему). Последнее является критической проблемой для долговременных приложений.

Часть I: Пулы памяти

Стандартный механизм управления памятью Ada-это пул памяти, определенный в системе пакет System.Storage_Pools. Пул памяти-это тип с тегами, который позволяет переопределить стандартный оператор " new " и связанную с ним процедуру Unchecked_Deallocation. Данный пул может быть связан с одним или несколькими типами доступа. Библиотека run-time GNAT поставляется с несколькими предопределенными пулами памяти, и Вы также можете создать свой собственный. Одной из основных реализаций, например, было бы инструментирование операций пула для печати трассировки на консоль при каждом выделении или освобождении памяти, а затем последующая обработка этих трассировок с помощью внешнего инструмента.

Это, конечно, немного утомительно, поэтому GNAT предоставляет пакет GNAT.Debug_Pools как гораздо более продвинутая реализация пула памяти, которая может в любое время во время выполнения программы отображать выделенную в данный момент память (наряду с обратным отслеживанием кода в точке выделения). Он также обнаруживает недопустимые ссылки на память (например, пытается разыменовать указатель на уже освобождённую память). Реализация эффективна и накладывает только очень небольшие накладные расходы на ваше приложение.

Вот краткий пример, демонстрирующий использование пулов отладки:

```
with GNAT.Debug_Pools;

package My_Package is
   Pool : GNAT.Debug_Pools.Debug_Pool;

   type Integer_Access is access Integer;
   for Integer_Access'Storage_Pool use Pool;
end My_Package;

with My_Package;
with Ada.Unchecked_Deallocation;
procedure Main is
   procedure Unchecked_Free is
      new Ada.Unchecked_Deallocation (Integer, Integer_Access);
   Ptr : Integer_Access;
begin
   Ptr := new Integer;
   Ptr.all := 1;
   Unchecked_Free (Ptr);
   Ptr.all := 2;  --  raises exception
```

```
end Main;
```

Переменная `My_Package.Pool` должна быть максимально общей для всех типов доступа. Нет необходимости в создании индивидуальной переменной для различных типов доступа.

Как отмечено в основной процедуре, последняя ссылка на Ptr недопустима и приведёт к исключению, вызванному из пула отладки (раньше, чем к некоторому ошибочному поведению, в зависимости от системы).

`GNAT.Debug_Pools` предоставляет различные подпрограммы для анализа текущего использования памяти, в частности общего объёма выделенной памяти, а также того, какая часть кода выполняла выделения из пула системы память. Обратные пути также полезны при анализе сценариев двойного освобождения, так как пул отладки показывает, где была выделена память и из какого фрагмента кода он был сначала освобождён.

Тем не менее, пулы отладки GNAT довольно тяжелы для размещения в существующем коде, так как вам нужно добавить "`for Type_name'Storage_pool Use Pool`" к каждому типу доступа, и нет механизма для определения одного пула хранения по умолчанию для всех типов. `GNAT.Debug_Pools` также может выдавать ложные предупреждения при разыменовании указателя на псевдонимы данных в стеке (который никогда не выделялся с помощью оператора "`new`", но был доступен через атрибут `'Access`).

В следующем выпуске Gem мы обсудим альтернативный подход к управлению и инструментированию динамического выделения и освобождения памяти, переопределив саму поддержку управления памятью низкого уровня.

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

Сведений об авторе нет.

Last Updated: 10/13/2017
Posted on: 1/11/2010

Обсуждение…

2 ответа на " Gem # 77: Куда делась моя память? (Where did my memory go?) (Часть 1)”

1. 11 января 2010 в 3:07 вечера

Дункан Сандис (Duncan Sands) сказал:

Я не уверен, что пример говорит что-либо об Debug_Pools:

вызов Unchecked_Deallocation установит значение Ptr равным null, поэтому

вы получили бы исключение здесь, использовали ли Вы Debug_Pools

или нет.

2. 11 января 2010 года в 3: 42

Эммануэль Бриот (Emmanuel Briot) сказал:

Да, это правда. На самом деле, пример может быть полезным

```
Ptr2: Integer_Access;

Ptr: = новое целое число;

Ptr2: = Ptr;

Unchecked_Free (Ptr);

Ptr2.all: = 2; - вызывает исключение
```