Gem #95: Динамический анализ стека в GNAT

Автор: Quentin Ochem, AdaCore

Краткое содержание: В Gem #95 описывается возможность GNAT динамически рассчитывать объем используемого задачей стека в процессе выполнения.

Давайте начнём…

Определение объёма пространства стека, выделяемого задачам, является распространённой проблемой управления памятью. В отсутствие поддержки инструментов, часто единственная информация, которая есть у разработчиков, - это выход сообщения `EXCEPTION_STACK_OVERFLOW`, когда их программа аварийно завершает работу. GNAT предлагает два основных способа для пользователей, чтобы получить информацию об использовании стека программы, - статический или динамический. Этот Gem посвящён второму способу, как получить данные использования стека динамически, то есть во время выполнения программы. Измерение статическим способом использования стека в программе, будет в следующей статье этой серии Gem.

Вычисление размера стека при завершении задачи

Начнём с простой программы, у которой есть задача, размер стека которой определяется во время выполнения:

```
procedure Main is

   task T is
      entry E (Size : Integer);
   end T;

   task body T is
   begin
      accept E (Size : Integer) do
         declare
            V : array (1 .. Size) of Integer := (others => 0);
         begin
            null;
         end;
      end E;
   end T;

begin
   T.E (500_000);
end Main;
```

Эта программа работает нормально, но каковы её требования к стеку? Есть ли вероятность, что, добавив новый код, который может потреблять дополнительный стек, мы попадем на ограничение? Давайте узнаем, скомпилировав это с помощью Stack instrumentation:

```
gnatmake main.adb -bargs -u10
```

Переключатель "-bargs-u10" заставляет "-u10" быть переданным к редактору связей GNAT, который позволит до десяти задач быть инструментованными и выведет их использование стека после завершения программы.

Скомпилированная таким образом, программа выдаёт следующую информацию:

```
  Index   | Task Name | Stack Size |      Stack usage
       1 | t         |    2097152 | 2008872 +/- 8188
```

Это означает, что из 2 097 152 байтов, доступных для стека задачи, в настоящее время программа использует 2 008 872.

Настройка размера стека

Наш стек выглядит достаточно полным, и, вероятно, разумно увеличить его размер, чтобы быть в безопасности, и избежать возможных исключений, когда программа будет дополнена новым кодом. Это можно сделать легко, используя `pragma Storage_Size`:

```
task T is
   pragma Storage_Size (3_000_000);
   entry E (Size : Integer);
end T;
```

Компиляция одной и той же программы с этими изменениями приводит к следующим цифрам:

```
  Index   | Task Name | Stack Size |      Stack usage
       1 | t         |    3000000 | 2008872 +/- 8188
```

Это гораздо более разумно.

Вычисление размера стека во время выполнения

Мы теперь собираемся создать новую версию задачи, которую можно вызвать многократно. Так как эта задача собирается жить дольше и сделать несколько вещей для различных клиентов, мы хотели бы зондировать задачу в разное время, а именно, каждый раз, когда вызывают точку взаимодействия задачи. Пакет среды выполнения `GNAT.Task_Stack_Usage` предоставляет средства инструментирования задачи. Давайте изменим тело задачи следующим образом:

```
task T is
   pragma Storage_Size (3_000_000);
   entry E (Size : Integer; Name : String);
end T;

task body T is
begin
   loop
      accept E (Size : Integer; Name : String) do
       declare
        V : array (1 .. Size) of Integer := (others => 0);
       begin
        Put_Line ("MAX USAGE OF T AFTER " & Name & ":"
         & Natural'Image (GNAT.Task_Stack_Usage.Get_Current_Task_Usage.Value));
       end;
      end E;
   end loop;
end T;
```

Обратите внимание на вызов `Get_Current_Task_Usage`, который вычисляет сумму стека, потребляемую до сих пор после каждого вызова `E`. Давайте теперь будем вызывать эту точку взаимодействия с задачей несколько раз, с разными значениями для размера массива:

```
   T3.E (5_000, "OP 1");
   T3.E (100_000, "OP 2");
```

```
   T3.E (20_000, "OP 3");
   T3.E (800_000, "OP 4");
```

Это приведёт к выводу следующих сообщений на консоль:

```
MAX USAGE OF T AFTER OP 1: 29392
MAX USAGE OF T AFTER OP 2: 409392
MAX USAGE OF T AFTER OP 3: 411204
raised STORAGE_ERROR : EXCEPTION_STACK_OVERFLOW
```

Обратите внимание, что размер стека вычисляется после каждого вызова, за исключением последнего вызова, что приводит к исключению. Также обратите внимание на интересный побочный эффект: «OP 3» должен занимать меньше места в стеке, чем «OP 2», поэтому мы обычно ожидаем, что число будет одинаковым. Что происходит, так это то, что в «OP 2» сначала вычисляется строка «`MAX USAGE OF T AFTER OP 2: 409392`», а затем вызывается `Put_Line`, которая сама потребляет некоторый стек, до уровня 411204 байта. Таким образом, 411204 на самом деле является максимальным объёмом пространства стека, используемого OP 2, хотя отображаемое значение меньше.

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

Quentin Ochem имеет опыт разработки программного обеспечения, специализирующийся на разработке программного обеспечения для критически важных приложений. Он имеет более чем 10-летний опыт разработки Ada. Сегодня он работает техническим менеджером AdaCore по проектам, связанным с авионикой, железнодорожной, космической и оборонной отраслями. Он также обучает стандарту DO-178B авионики в университете EPITA в Париже.

Last Updated: 10/13/2017
Posted on: 11/22/2010

Обсуждение…